Algorithms for Optimization Problems with Fractional Resources

Marco Casazza

UNIVERSITÀ DEGLI STUDI DI MILANO
Dipartimento di Informatica

UNIVERSITÉ PARIS 13
Laboratoire d'Informatique Paris Nord

Ph.D. Thesis:

ALGORITHMS FOR OPTIMIZATION PROBLEMS WITH FRACTIONAL RESOURCES

Candidate:
MARCO CASAZZA

Tutors:
DR. ALBERTO CESELLI
PROF. ROBERTO WOLFLER CALVO

Coordinator:
PROF. ERNESTO DAMIANI

Thesis defended on 26 February 2016

Examination committee members:
Claudio Ferretti
Lucas Létocart

Referees:
Andrea Lodi
Dominique Feillet
Manuel Iori
Stefan Irnich

XXVIII cycle
A.A. 2014/2015

ABSTRACT

In this thesis we consider a class of optimization problems having a distinctive feature: both discrete and continuous decisions need to be taken simultaneously. These problems arise in many practical applications, for example broadband telecommunications and green transportation problems, where resources are available, that can be fractionally consumed or assigned. These problems are proven of being harder than their purely discrete counterpart. We propose effective methodologies to tackle them. Our approach is to consider variants of classical combinatorial optimization problems belonging to three domains: packing, routing, and integrated routing / packing. Our results suggest that indeed effective approaches exist, reducing the computational effort required for solving the problem. Mostly, they are based on exploiting the structure of optimal solutions to reduce the search space.

RIASSUNTO

In questa tesi affrontiamo una classe di problemi di ottimizzazione con una caratteristica in comune: sia le decisioni discrete che quelle continue devono essere prese simultaneamente. Questi problemi emergono in molti campi, come ad esempio le nelle telecomunicazioni a banda larga e in problemi di trasporto ecologico, dove le risorse disponibili possono essere consumate o assegnate in modo frazionario. Questi problemi sono generalmente più difficili da risolvere rispetto alla loro controparte puramente combinatoria. Noi proponiamo metodologie efficaci per affrontarli. Con il nostro approccio consideriamo varianti di problemi classici nel campo dell'ottimizzazione combinatoria che appartengono a tre domini: impaccamento, instradamento e instradamento / impaccamento integrati. I nostri risultati suggeriscono l'esistenza di approcci efficienti che riducono lo sforzo computazionale necessario per risolvere questi problemi. Nella maggior parte dei casi, tali approcci sono basati sullo sfruttamento di particolari proprietà della struttura delle soluzioni ottime in modo da ridurre lo spazio di ricerca.

RÉSUMÉ COURT

Dans cette thèse nous considérons une classe de problèmes d'optimisation ayant une particularité : des décisions à la fois discrètes et continues doivent être prises simultanément. Ces problèmes se posent dans de nombreuses applications pratiques, comme par exemple dans les réseaux de télécommunications à large bande passante et dans les problèmes de transport écologique, où les ressources disponibles peuvent être très légèrement consommées ou réparties. Ces problèmes se sont avérés être plus difficiles à résoudre que leurs homologues purement discrets. Des méthodes efficaces pour la résolution de ces problèmes sont proposées dans cette thèse. Notre approche est de prendre en compte des variantes de problèmes classiques d'optimisation combinatoire appartenant à trois domaines : packing, routage et routage / packing intégré. Les résultats obtenus suggèrent l'existence de méthodes efficaces, réduisant l'effort de calcul nécessaire pour résoudre ce type de problème. La plupart du temps, ces méthodes sont basées sur l'exploitation de la structure des solutions optimales pour réduire l'espace de recherche.

CONTENTS

1

INTRODUCTION

Logistics has always been a benchmark for combinatorial optimization methodologies, as practitioners are traditionally familiar with the competitive advantage granted by optimized systems. As an example, the Vehicle Routing Problem (VRP) [29], where a fleet of vehicles must deliver goods to customers in a network, is popular since decades for operational planning. Indeed, as more problems are understood from a computational point of view, more details can be included in state-of-the art models to improve their representation of reality, driving research for new solution methods in a virtuous cycle.

With this perspective, in the last decade many studies such as [73] and [6], have proved that problems having both combinatorial and continuous decisions are a new frontier in optimization. For instance, let us consider the Split Delivery Vehicle Routing Problem (SDVRP), that is a relaxation of the VRP in which splits are allowed. In such a problem, each customer may be visited more than once and its demand can be fractionally assigned to different vehicles. In [6], the authors proved that cost savings by allowing splits may reach 50%.

From a methodological point of view, there is currently a strong concern in tackling optimization problems in which combinatorial and continuous decisions must be taken at the same time. It is the case, for instance, of the Green Vehicle Routing Problem (GVRP)[66], where it is required to route a fleet of electric vehicles on a network in order to serve customer demands. Indeed the limited battery capacity of these vehicles requires to visit recharging station during delivery tour, and therefore a recharge planning is required to avoid running out of battery. However, charging an electric vehicle requires a certain amount of time that is not negligible. Therefore, when driving time constraints or time windows are imposed, partial charging must be considered as an option into the problem, allowing a vehicle to recharge a fraction of its capacity, trading battery for time.

Still concerning vehicles with alternative fuels, the increasing of vehicles running on hybrid technologies cause to reconsider well-known problems like the Shortest Path Problem (SPP). In fact, in addition to the choice of a path, considering fuels with different properties im-

poses to choose the fractions of each fuel used to cover a given distance.

Unfortunately, state-of-the-art methodologies and general purpose solvers still fail to solve efficiently the class of problems described above.

The aim of my thesis is to investigate common approaches to solve efficiently different categories of optimization problems with fractional resources. To achieve such a goal we devise new methods exploiting properties on the structure of solutions, reducing the search space, and improving the computational performances of ad-hoc algorithms. In particular, we focused on decomposition methods and column generation techniques as these allow to embed structural properties of the solutions into simpler and more manageable sub-problems.

We focus on variants with fractional resources of well-known problems in literature, such as packing problems and routing problems. These are well-suited for decomposition methods, as evidenced for example in [77] and [19], and are therefore a good starting point for our research.

In the first part, we tackle the fractional variant of the Bin Packing Problem (BPP), where a set of items must be packed into a set of bins. In such variant splits are allowed at a cost, and depending on the practical application, multiple versions of such a problem have been discussed in the literature. In Chapter 3 we propose an exact approach, discussing properties on the structure of the optimal solutions and devising new mathematical programming models. We also exploit decomposition methods to obtain an extended formulation, and column generation techniques to implement a branch-and-price algorithm performing substantially better than general purpose solvers. In Chapter 4 we then extend properties to restrict the problem to its pure combinatorial core, embedding the split component into the structure of the solutions, and devise a new algorithm still based on branch-and-price. Such approach is both flexible and efficient: in fact, in Chapter 5 we extend the underlying theory to different variants of the problem, obtaining a framework that is able to solve instances of an order of magnitude bigger, in computational time of orders of magnitude smaller than the previous ones. In Chapter 6 we introduce a new variant of the problem and show a reduction method to a well-known problem.

In the second part, we tackle a generalization of the Split Delivery Routing Problem that arises in bike sharing systems, where bikes are

taken from public stations at a cost, and used for a limited amount of time until they are taken back to another possibly different station. In Chapter 7 we propose an exact algorithm based on branch-and-price, that exploits properties on the structure of the solutions to simplify the resolution of the pricing problem. This approach allows to obtain optimal solution orders of magnitude faster than competitors.

Depending on the nature of the problem at hand and the kind of resources, diverse techniques may be needed. In Appendix A we include as a representative example an additional selected contribution in which we propose methodologies to solve a last mile home delivery problem. In such a problem, a store has to schedule deliveries of products to its customers, defining a feasible time window for each delivery or rejecting the request if none is available. Indeed it is not possible to know all the requests a priori, and therefore, whenever a new customer arrives, all the decisions must be taken evaluating only the partial schedule. To obtain an high quality of service, the store needs to satisfy the highest number of demands, eventually suggesting small variations to the time windows proposed by the customer. Indeed, because of the online nature of the problem it was inappropriate to design offline deterministic optimization procedures. Instead, we propose new policies to define variations of the time windows of customers, with different measurements of quality. In that case, time is a resource that needs to be fractionally assigned to customers.

In summary, we can draw a coherent conclusion that general purpose solvers are still not suitable to solve such problems, and specific techniques are needed to tackle problems with fractional resources.

LIST OF PUBLICATIONS. Contents extracted from this thesis have already been published in international journals or presented to international conferences.

Publications in international journal or conferences with peer-reviewed published proceedings:

- M. Casazza and A. Ceselli. Mathematical Programming Algorithms for Bin Packing Problems with Item Fragmentation. *Computers & Operations Research*, 2014
- M. Casazza, A. Ceselli and L. Létocart. Optimizing time slot allocation in single operator home delivery problems. *GOR 2014 – Operations Research Proceedings*, 2014

Publications in international conferences and workshops:

- M. Casazza and A. Ceselli. Improved algorithms for Bin Packing Problems with Item Fragmentation. *EURO 2013*
- M. Casazza. Optimization algorithms for packing problems with fragmentation. *ELA Doctorate Workshop*
- M. Casazza and A. Ceselli. An exact algorithm for bin packing problems with item fragmentation. *ODYSSEUS 2012*

Papers under review and technical reports:

- M. Casazza and A. Ceselli. Exactly solving packing problems with fragmentation. Submitted to *Computers & Operations Research*, technical report available on *Optimization Online*

Part I

PACKING PROBLEMS

2

BIN PACKING WITH ITEM FRAGMENTATION

Packing problems represent one of the most fundamental, and still lively, research fields in combinatorial optimization.

In the classical Bin Packing Problem (BPP), a set of bins of limited capacity and a set of items of known weight are given. The task is to pack items into the minimum number of bins without exceeding their capacity. One of the key feature of the problem is that items cannot be split, and each one must be fully assigned to a single bin.

Packing problems are of particular relevance in terms of direct applicability in real life problem: in logistics, items often correspond to transportation requests placed by customers, while bins correspond to vehicles trailers. A BPP arises whenever a tactical fleet sizing problem has to be faced [29]. BPPs also find direct application in telecommunication planning problems, where items correspond to data transfer requests, bins correspond to transmission channels, and network dimensioning must be carried out [70].

From a methodological point of view, BPP is well-known and some exact approaches exist [33] that solve instances with up to hundreds of items in few minutes. In particular, branch-and-price approaches are well-suited to tackle such problem: BPP is usually modelled as a set partitioning like problem in which a set of bin patterns is given, with the objective of minimize the number of patterns selected [57]. Indeed, the set of patterns grows exponentially in the number of items and column generation techniques are required to solve the continuous relaxation of the model. The bound given by this relaxation is usually very tight. In [75] the author proposes an arc-flow formulation that is solved by means of branch-and-price-and-cut. In such formulation, flow paths corresponding to bin patterns are generated, and the branching strategy requires less modifications to the structure of the pricing problem. A different approach is studied in [13], where cutting planes are added to the model without considering the eventual destruction of the structure of the pricing problem, which is solved by means of a branch-and-bound method. Furthermore, dual cuts were proposed in [78], [14], and [26] to accelerate the column generation process.

Of course, stronger market competition and new technologies push for more elaborated methods, asking for more sophisticated models that may be able to capture more application details and provide more optimization power. This is the case, for instance, of Split Delivery Vehicle Routing Problems (SDVRP) in transportation [4], where multiple visits are allowed to each customer, in order to split its transportation request between multiple vehicles; however, multiple visits are known to reduce the level of service perceived by the customer.

Another example is the Fully Optical Network Planning Problem (FONPP)[68]. Such problem arises on fully optical Petaweb networks, that are high rate transmission networks in which communication between *edge nodes* is granted by optical switch called *core nodes*. An example of Petaweb network is pictured in Figure 1. To perform a

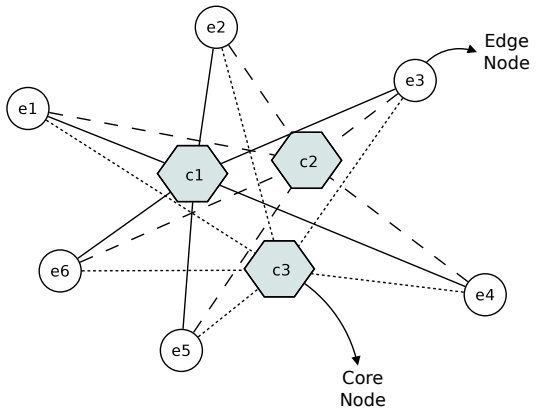

Figure 1: Example of PetaWeb network with 6 edge nodes and 3 core nodes.

transmission it is required to select a communication channel, that is a triple of physical channel, frequency of the signal, and time slot. In order to route all the transmissions, the problem can be formulated as a packing problem in which transmissions must be assigned to different communication channels without exceeding their capacity. Indeed, communications can be split among different channels, but each split induce a cost due to the electrical energy required to split the optical signal.

The BPP with Item Fragmentation (BPPIF) has actually been introduced in the literature to suit these needs, allowing items to be split at a price. The BPPIF is known to be NP-hard and was first introduced in [56].

Although there is a vast literature on the BPP, the same does not go for the BPPIF. In [56] the authors present a worst case analysis of the problem and two approximation algorithms: the *Next-Fit with Item Fragmentation* and the *Best-Fit with Item Fragmentation*. Both algorithms are adapted for the BPPIF by extending well-known approximation algorithm for the BPP. In [69] the authors present dual asymptotic fully polynomial time approximation schemes that represent the state-of-the-art in approximately solving BPPIFs. Furthermore, they introduce the concept of *primitive structure*, and show that for a certain variant of the BPPIF it always exists an optimal solution that is primitive. the authors studied its computational complexity and discussed the approximation properties of traditional BPP heuristics. Similar models have been introduced in the context of memory allocation problems by [25] and considered in [40]: BPPs are presented in which items can be split, but each bin can contain at most k item fragments; the theoretical complexity is discussed for different values of k, and simple approximation algorithms are given. Such results have been refined in [41], where the authors provide efficient polynomial-time approximation schemes, and consider also dual approximation schemes. A two-dimensional packing problem in which items can be split is addressed in [54], where the authors study its computational complexity and propose efficient heuristics.

Many BPPIF variants have been discussed in the literature. In fact, for what concerns capacity consumption, two versions of the BPPIF can be found: the simpler BPPIF with Size Preserving Fragmentation (BPPSPF) in which the sum of weights of fragments of an item is constant, and the BPP with Size Increasing Fragmentation (BPPSIF), in which an overhead is attached to each fragment whenever it is packed. Instead, for what concerns the objective of the optimization, the BPPIF arises in the literature in both fragmentation-minimization form as in [68], or in bin-minimization form as in [69]. In the former, a finite set of bins is given and the task is to minimize the overall number of fragmentations. In the latter, an upper limit on the total number of fragmentations is imposed, that is the available budget, and a solution minimizing the number of used bins is required. However, to the best of our knowledge, no exact approach has been proposed so far for any of the variants.

In this thesis we investigate on BPPIF properties and devise different models and methods to solve such problems. These yield compact models that avoid the use of fractional variables. Exploiting our new

tools, we also present a computational comparison on BPPIF models, assessing the impact of overhead handling on solution values and computing hardness.

We also introduce the fragmented-item-minimization variant of the BPPIF. This variant arises as an operative problem in Split Delivery VRP, in which a limited number of vehicles is provided and the task is to minimize the number of customers visited multiple times. We propose a reduction of the Size Preserving variant to a pure combinatorial problem, solving approximatively 80% of the instances to optimality.

In Chapter 3 we describe the fragmentation-minimization BPPIF, formalize the problem by means of a compact mathematical programming model, and present our resolution method. We then devise a different resolution approach in Chapter 4, by exploiting an improved formulation. We then extend such formulation and the underlying theory to the bin-minimization variant in Chapter 5. In Chapter 6 we introduce the new fragmented-item-minimization variant and show our reduction method. For all variants we also report results of the computational analysis to prove the effectiveness of the methods. For the sake of readability, we summarize the content of the chapters on the BPPIF as follows:

	fragmentation minimization	bin minimization	fragmented item minimization
Size Preserving	fm-BPPSPF Chapter 3 and 4	bm-BPPSPF Chapter 5	fim-BPPSPF Chapter 6
Size Increasing	fm-BPPSIF Chapter 4	bm-BPPSIF Chapter 5	-

MAIN NOVEL CONTRIBUTIONS. In this part of the thesis we address a different version of the BPPIF, contributing to the state of the art on the methodologies to solve problems with fractional resources to optimality. We remark that no exact approach was studied in literature, and therefore our algorithms are already a novelty. In particular:

(a) we introduce new variants of the BPPIF that have not been addressed in literature yet;

(b) we study theoretical properties that hold for some variants of BPPIF;

(c) we exploit such properties to present new mathematical programming models to both describe and solve BPPIFs;

(d) we exploit decomposition methods to obtain extended formulations and we make use of column generation techniques to solve their continuous relaxations;

(e) we devise feasible dual cuts to improve column generation stabilization;

(f) we develop branch-and-price frameworks to solve BPPIFs, including new branching strategies, primal heuristics, and feasibility detection procedures;

(g) we reduce a variant of the BPPIF to a well-known pure combinatorial problem;

(h) we study the performances of our algorithms by means of an extensive experimental campaign.

3

FRAGMENTATION MINIMIZATION BPPSPF

We begin by formalizing the problem and by presenting some structural properties of the fm-BPPSPF. The key result is that the search for optimal solutions can be pursued, without loss of optimization power, by considering only *primitive solutions*, that are solutions having a particular structure, and thereby reducing the search space.

3.1 PROBLEM DEFINITION

It is given a set of items I and a set of bins B. Let w_i be the weight of each item $i \in I$ and let C be the capacity of each bin $j \in B$. The fm-BPPSPF can be stated as the problem of packing all the items in I into the bins of B. Each item can be split into *fragments* and fractionally assigned to different bins. The aim is to perform the packing in such a way that the sum of the weights of the (fragments of) items packed into a single bin does not exceed the capacity C, minimizing the overall number of fragmentations.

Definition 3.1.1. *Formally, a solution to the fm-BPPSPF is a function* $\phi : I \times B \to [0, 1]$ *indicating the fraction of each item* i *packed into each bin* j.

Let us define as W_j the amount of space used in bin j, that is

$$W_j = \sum_{i \in I} w_i \cdot \phi(i, j)$$

and as $\bar{W}_j$ the residual space

$$\bar{W}_j = C - W_j$$

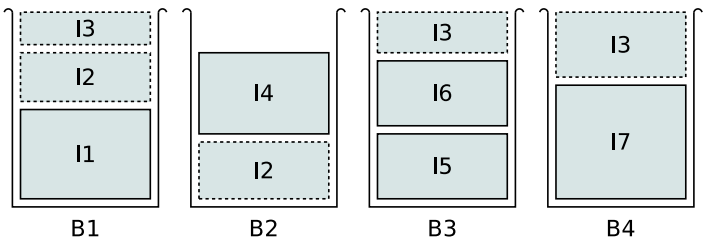

Figure 2: Instance with $|I| = 7$ and $|B| = 4$. The fragmented items are 2 and 3

A solution ϕ is feasible if

(a) $\bar{W}_j \geqslant 0$ for each $j \in B$,

(b) $\sum_{j \in B} \phi(i,j) = 1$ for each $i \in I$.

Definition 3.1.2. *The cost of a packing ϕ, is its total number $\mathcal{F}$ of fragmentations, can be stated as follows:*

$$\mathcal{F} = |\{(i,j) \in I \times B : \phi(i,j) > 0\}| - |I| \tag{3.1.1}$$

Indeed, to find how many times each item $i \in I$ is fragmented in a solution ϕ, we count how many fragments are contained in the solution and subtract one

$$|\{j \in B : \phi(i,j) > 0\}| - 1;$$

to obtain the total number of fragmentations we sum over all items

$$\sum_{i \in I} |\{j \in B : \phi(i,j) > 0\}| - \sum_{i \in I} 1$$

obtaining expression (3.1.1). It also follows

Observation 3.1.1. *Each fm-BPPSPF solution minimizing the overall number of fragmentations minimizes the overall number of fragments as well.*

In fact, $|I|$ is a constant that does not affect the optimization process. A sample instance, and solution in which two items are fragmented, is depicted in Figure 2.

3.1.1 *Structure of a solution*

In principle, fm-BPPSPF solutions have a large degree of freedom, potentially leading to symmetry problems during search. However, we show that representatives of optimal solutions exist, having particular structure. We first observe the following.

Observation 3.1.2. *If an item $i \in I$ has weight $w_i > C$, then an optimal fm-BPPSPF solution can always be obtained by assigning a fragment C/w_i to a single bin, completely filling its capacity, and by considering a residual instance having $|B| - 1$ bins, and in which w_i is reduced by C.*

Proof. Suppose to have a solution in which such an item i is fragmented among a set of bins $\tilde{B}$, without completely filling any of them.

Now, consider the bins in $\tilde{B}$ in random order $k_1, k_2 \ldots$. Keep bin k_1 as an accumulator, and starting from $j = 2$, check the following terminating condition: $C - w_i \cdot \phi(i, k_1) \leqslant w_i \cdot \phi(i, k_j)$.

(a) If this condition holds, then swap each item fragment placed in bin k_1, besides that of item i, with a portion of the fragment of i placed in bin k_j having same overall size: this operation keeps the overall number of fragmentations unchanged, and completely fills the capacity of the bin k_1 with a unique fragment of i.

(b) Otherwise, select a subset of fragments in bin k_1 whose overall size equals $w_i \cdot \phi(i, k_j)$, without including fragments of i, maybe fragmenting a single additional item. Swap this subset with the fragment of i in bin k_j. As before, this operation does not increase the overall number of fragmentations; at the same time, the capacity of the bin k_1 used by a fragment of i increases.

It is easy to check that the terminating condition necessarily becomes true as j reaches $|\tilde{B}|$, and that performing operation (a) yields our claim. □

Therefore, w.l.o.g. we assume $w_i < C$ for all $i \in I$, as otherwise the instance can be simplified by preprocessing. We then introduce the following two particular cases of packings.

Definition 3.1.3. *We define as* minimal w.r.t. a given number of fragmentation $\mathcal{F}$*, a packing performing at most $\mathcal{F}$ fragmentations and minimizing the number of bins used.*

Definition 3.1.4. *We define as* minimal *a packing using a minimum number $\mathcal{F}$ of fragmentations and that is minimal w.r.t. $\mathcal{F}$.*

Definition 3.1.5 ([69])**.** *A packing is* primitive *if (a) it is feasible (b) each bin contains at most two fragmented items, (c) each item is fragmented at most once.*

An example of primitive packing is depicted in Figure 4.

The notion of primitive packings can be easily represented through the construction of a *Bin Packing Graph* (BPG), having one vertex for each bin and one edge between each pair of bins sharing a fragmented item (see Figure 3); in a primitive packing, the BPG is a collection of paths or cycles (see Figure 4). We also observe the following:

Observation 3.1.3. *Each primitive packing whose corresponding BPG contains cycles can be transformed into a primitive packing whose corresponding BPG is composed by paths only.*

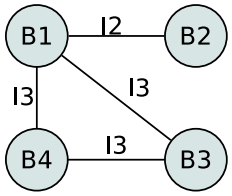

Figure 3: Example of BPG of the non-primitive solution in Figure 2. An edge connects each pair of bins sharing a fragmented item. Item 3 is shared between bins 1, 3, and 4 inducing a cycle in the BPG.

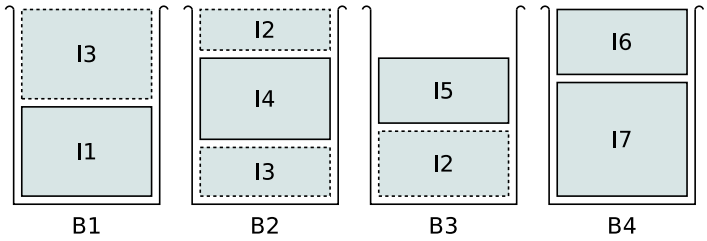

Figure 4: Example of primitive packing

Proof. In order to remove a cycle p corresponding to the set of bins K_p, it is enough to remove all items from the bins of the cycle, and greedily pack them back into the bins using the *Next-Fit with Item Fragmentation (NF_f)* algorithm described in [56], as follows:

(a) open an empty bin;

(b) choose an item $i \in I$ and fully pack it without fragmentations into the current bin;

(c) if no item $i \in I$ fits into the current bin, split it in two fragments. Pack the first one filling the residual capacity, and close the current bin. Open a new bin and pack the second fragment;

(d) remove the selected item from I and go to (b) until I is empty.

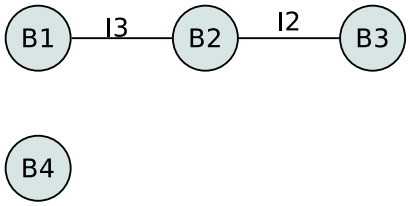

Figure 5: Example of BPG of the primitive solution in Figure 4.

NF_f algorithm is described also in Pseudocode 3.1.1. As proved in [56], NF_f produces a solution with at most $|K_p| - 1$ fragmentations, that is a path in the BPG. □

```
function NFF(I, B, w, C)
    S ← ∅
    C_residual ← C
    b ← 1
    for all i ∈ I do
        if C_residual ⩾ w_i then
            S ← {(i, b, 1)}
            C_residual ← C_residual − w_i
        else if b < |B| then
            S ← {(i, b, C_residual/w_i), (i, b + 1, (w_i − C_residual)/w_i)}
            C_residual ← C − (w_i − C_residual)
            b ← b + 1
        end if
    end for
    return S
end function
```

Pseudocode 3.1.1: Pseudocode of the Next-Fit with Item Fragmentation algorithm.

Definition 3.1.6. *A chain is a sequence of bins corresponding to a path of the BPG.*

From now on, we assume that each primitive solution is a collection of chains. We say that an item belongs to a chain or that it is packed into a chain, if it is packed into a bin of such chain.

Different properties of primitive solutions are discussed in the literature; in particular [69]:

Remark 3.1.1. *In the bm-BPPSPF, there always exists an optimal solution which is primitive.*

Now, still w.l.o.g., we can restrict the search for optimal fm-BPPSPF solutions to the following subset of feasible solutions.

Theorem 3.1.1. *For each instance of fm-BPPSPF it always exists an optimal packing which is both minimal and primitive.*

Proof. Let ϕ^* be an optimal fm-BPPSPF packing, and let $\mathcal{F}^*$ be the number of fragmentations in ϕ^*; let β^* be an optimal solution of a bm-BPPSPF on the same instance, in which the upper bound on the total number of fragmentations is set to $\mathcal{F}^*$. According to Remark 3.1.1, we

can assume β^* to be primitive; we also observe that the number of fragmentations in β^* must be exactly $\mathcal{F}^*$, as otherwise ϕ^* would not be optimal for fm-BPPSPF. Now, the packing ϕ^* is feasible for bm-BPPSPF as well, and therefore the number of bins used in β^* is less than or equal to those used in ϕ^*. At the same time β^* (a) is feasible for fm-BPPSPF, as it is using at most the same number of bins (b) it is optimal, since it contains $\mathcal{F}^*$ fragmentations (c) it is primitive and (d) it is minimal according to definition 3.1.3. □

As a side effect, the constructive proof of Theorem 3.1.1 is able to build the primitive counterpart of any fm-BPPSPF solution, and therefore

Corollary 3.1.2. *For each instance of fm-BPPSPF there always exists an optimal packing which is primitive.*

We now focus on the inner structure of chains in the BPG. Let K_p be a set of bins corresponding to vertices in the same chain p in the BPG. Let I_p be the set of items packed into bins in K_p. We recall that following the definition of BPG, no item can be packed into bins corresponding to different chains.

Theorem 3.1.3. *Let $\tilde{I}_p$ be a set containing the $|K_p| - 1$ items of I_p having maximum weight. It is always possible to pack all the items in I_p into the bins of K_p by fragmenting only items in $\tilde{I}_p$.*

Proof. Such a packing can be obtained with the following variant of NF_f:

(a) open an empty bin;

(b) iteratively, choose an item $i \in I_p \setminus \tilde{I}_p$, pack it without fragmentations into the current bin, and remove it from I_p;

(c) if no item $i \in I_p \setminus \tilde{I}_p$ fits into the current bin, choose one of the items in $\tilde{I}_p$, split it in two fragments, the first one filling the residual capacity of the current bin; remove such an item from $\tilde{I}_p$; close the current bin and open a new one, packing the second fragment;

(d) go to (b) until there are no items left in $I_p \setminus \tilde{I}_p$;

(e) pack the remaining items in $\tilde{I}_p$ sequentially using the original version of NF_f.

It is easy to check that (1) the packing produced by the procedure above is feasible, as the capacity of each bin is never exceeded, and

no bin is opened until the previous one is completely full, (2) at most $|K_p| - 1$ items are fragmented (those in $\tilde{I}_p$). □

Our variant of NF_f can be seen as a way of finding a proper sorting of items during the packing process, and indeed our result confirms that reported in [56], where the authors prove that NF_f produces a feasible packing with at most $|K_p| - 1$ fragmentations for *any* sorting of the items in I_p.

Summarizing, we can state the following

Theorem 3.1.4. *In optimizing the fm-BPPSPF, we can restrict the search to solutions being both minimal and primitive, and in which the fragmented items are the largest in each chain, without losing optimization potential.*

Proof. The result can be directly drawn from theorems 3.1.1 and 3.1.3. □

3.2 MODELING

We now focus on methods for obtaining good lower bounds. Hence, we introduce the following mixed integer linear programming formulation for the fm-BPPSPF:

$$\min \quad \sum_{\substack{i\in I\\ j\in B}} z_{ij} \tag{3.2.1}$$

$$\text{s.t.} \quad \sum_{j\in B} x_{ij} = 1 \qquad \forall i\in I \tag{3.2.2}$$

$$\sum_{i\in I} w_i \cdot x_{ij} \leqslant C \qquad \forall j\in B \tag{3.2.3}$$

$$x_{ij} \leqslant z_{ij} \qquad \substack{\forall i\in I\\ \forall j\in B} \tag{3.2.4}$$

$$0 \leqslant x_{ij} \leqslant 1 \qquad \substack{\forall i\in I\\ \forall j\in B} \tag{3.2.5}$$

$$z_{ij} \in \{0,1\} \qquad \substack{\forall i\in I\\ \forall j\in B} \tag{3.2.6}$$

where each continuous variable x_{ij} represents the fraction of item i packed into bin j, and each binary variable z_{ij} is 1 if any fragment of item i is packed into bin j, 0 otherwise.

Constraints (3.2.2) impose that each item is fully assigned to bins, while constraints (3.2.3) ensure that the capacity of each bin is not exceeded. Constraints (3.2.4) enforce consistency between variables, that is no fragment of each item i is inserted into bin j unless z_{ij} is set to

1. According to definition 3.1.2, the objective function (3.2.1) equivalently minimizes both the number of fragments and the number of fragmentations.

It is easy to check that the lower bound given by the continuous relaxation of the previous model is not significant, as an optimal fractional solution of value 0 can always be found by setting $z_{ij} = x_{ij} = 1/|B|$ for each $i \in I, j \in B$. Therefore, we propose a reformulation of the problem obtained through Dantzig-Wolfe decomposition [30].

In particular, let $x_j = (x_{1k}, \dots, x_{|I|j})$, $z_j = (z_{1k}, \dots, z_{|I|j})$ and $w = (w_1, \dots, w_{|I|})$. Let, for each j in B,

$$\Omega_j = \left\{ (x_j, z_j) \in \mathbb{R}_+^{|I|} \times \mathbb{B}^{|I|} \mid w^T x_j \leqslant C \wedge z_j - x_j \geqslant 0 \right\}$$

be the set of feasible integer points with respect to constraints (3.2.3) – (3.2.6); we relax integrality conditions, but replace each Ω_j with the convex hull of its L_j extreme integer points

$$\Gamma_j = \left\{ (\bar{x}_j^1, \bar{z}_j^1), \dots, (\bar{x}_j^{L_j}, \bar{z}_j^{L_j}) \right\};$$

that is we impose

$$(x_j, z_j) = \sum_{l \in \Gamma_j} (\bar{x}_j^l, \bar{z}_j^l) \cdot y_j^l \tag{3.2.7}$$

with $y_j^l \geqslant 0$ for each $j \in B$ and $l \in \Gamma_j$, and $\sum_{l \in \Gamma_j} y_j^l = 1$ for each $j \in B$.

The model obtained by replacing in the continuous relaxations of formulation (3.2.1) – (3.2.6) the vectors (x_j, z_j) as indicated in (3.2.7), and by making explicit the vector indices is

$$\begin{aligned}
\min \quad & \sum_{j \in B} \sum_{l \in \Gamma_j} \sum_{i \in I} \bar{z}_{i,j}^l \cdot y_j^l & & \\
\text{s.t.} \quad & \sum_{j \in B} \sum_{l \in \Gamma_j} \bar{x}_{i,j}^l \cdot y_j^l = 1 & \forall i \in I & \quad (3.2.8) \\
& \sum_{l \in \Gamma_j} y_j^l = 1 & \forall j \in B & \quad (3.2.9) \\
& y_j^l \geqslant 0 & \substack{\forall j \in B \\ \forall l \in \Gamma_j} & \quad (3.2.10)
\end{aligned}$$

We also observe that since bins are identical, so are the sets Γ_j. Therefore, we consider a single representative $\Gamma = \bigcup_{j \in B} \Gamma_j$, and aggregate constraints (3.2.9) as

$$\sum_{l \in \Gamma} y^l \leqslant |B|. \tag{3.2.11}$$

Furthermore, exploiting Theorem 3.1.1, for each item $i \in I$ we can add an additional constraint

$$\sum_{l \in \Gamma} \bar{z}_i^l \cdot y^l \leqslant 2$$

which limits the number of fragments for each item, obtaining a Master Problem (MP), that in canonical form reads as follows:

$$\min \quad \sum_{\substack{l \in \Gamma \\ i \in I}} \bar{z}_i^l \cdot y^l \tag{3.2.12}$$

$$\text{s.t.} \quad \sum_{l \in \Gamma} \bar{x}_i^l \cdot y^l \geqslant 1 \qquad \forall i \in I \tag{3.2.13}$$

$$-\sum_{l \in \Gamma} \bar{z}_i^l \cdot y^l \geqslant -2 \qquad \forall i \in I \tag{3.2.14}$$

$$-\sum_{l \in \Gamma} y^l \geqslant -|B| \tag{3.2.15}$$

$$y^l \geqslant 0 \qquad \forall l \in \Gamma \tag{3.2.16}$$

Observation 3.2.1. *The lower bound provided by the MP is at least as tight as that given by the continuous relaxation of the original model.*

The observation directly follows from the Dantzig-Wolfe decomposition principle. As discussed in Section 3.5, we found such a lower bound to be good also from an experimental point of view.

3.3 ALGORITHMS

Straightly solving the MP would be impractical, as it would require to consider a tableau with $|\Gamma|$ columns; therefore, we recur to column generation techniques: we start with a Restricted Master Problem (RMP) involving a small subset of columns (see Subsection 3.3.1), we solve it to optimality and we use dual information to search for variables having negative reduced cost (see Subsection 3.3.2). If any such a variable is found, it is added to the RMP and the column generation process is repeated, otherwise the optimal RMP solution is optimal for the MP as well, and therefore the corresponding value is retained as a valid lower bound for the fm-BPPSPF. If the final RMP solution is integer, then it is also optimal for the fm-BPPSPF; otherwise, in order to find a proven global optimum, we enter a search tree by performing

branching operations (see Subsection 3.3.3) and feasibility checks (see Subsection 3.3.4).

3.3.1 *Initialization*

In order to ensure feasibility, we initially populate the RMP with $|I|$ *dummy* columns of very high cost, having coefficient one corresponding to constraints (3.2.13) and coefficient zero elsewhere; we also include a set of columns generated by two heuristics described in Section 3.4.2. Furthermore, in each inner node of the search tree, we keep in the RMP all the columns generated so far that are compatible with the branching decisions. In our experiments this simple strategy turned out to be enough to avoid heading-in effects [80].

3.3.2 *Pricing problem*

Let λ, μ and η be the vectors of non negative dual variables corresponding to constraints (3.2.13), (3.2.14) and (3.2.15); the reduced cost of each variable y^l is

$$\sum_{i\in I} \bar{z}_i^l - \sum_{i\in I} \lambda_i \bar{x}_i^l + \sum_{i\in I} \mu_i \bar{z}_i^l + \eta$$

The pricing problem, that is the problem of finding a most negative reduced cost column, can be stated as follows:

$$\min \quad -\sum_{i\in I} \lambda_i \cdot \bar{x}_i^l + \sum_{i\in I} (\mu_i + 1) \cdot \bar{z}_i^l + \eta \tag{3.3.1}$$

$$\text{s.t.} \quad \sum_{i\in I} w_i \cdot \bar{x}_i^l \leqslant C \tag{3.3.2}$$

$$\bar{x}_i^l \leqslant \bar{z}_i^l \qquad \forall i\in I \tag{3.3.3}$$

$$0 \leqslant \bar{x}_i^l \qquad \forall i\in I \tag{3.3.4}$$

$$\bar{z}_i^l \in \mathbb{B} \qquad \forall i\in I \tag{3.3.5}$$

Let us state the objective function of the pricing problem (3.3.1) in maximization form

$$\max \sum_{i\in I} \lambda_i \cdot \bar{x}_i^l - \sum_{i\in I} (\mu_i + 1) \cdot \bar{z}_i^l - \eta. \tag{3.3.6}$$

Since η is a constant, it does not affect the optimization process; instead we can consider λ_i to be the prize obtained by fully packing the

item i, and $\mu_i + 1$ to be the penalty for including any fragment of item i in the current column. Intuitively, the pricing problem aims at finding a set of fragments of maximum value, that does not exceed the capacity of a bin; the value of a set of fragments is computed as the difference between the collected prizes and the corresponding penalties. We indicate such a variant of the $0-1$ Knapsack Problem (KP) [55], formulated as (3.3.6), (3.3.2) – (3.3.5), as *Fractional KP with Penalties* (FKPP). We observe that the FKPP cannot be solved to optimality by means of the Dantzig algorithm for the continuous KP [31]. In fact, let us consider an instance with two items p and q, with weights $w_p = w_q = 1$, profits $\lambda_p = 10$ and $\lambda_q = 5$, and penalties $\mu_p = 9$ and $\mu_q = 0$. Given a knapsack with capacity $C = 1$, the Dantzig algorithm would start fully packing item p, obtaining a solution with value 0, because of the high penalty μ_p. Instead, packing q first would lead to a solution with value 4, which is optimal.

However, we prove that optimal FKPP solutions exist, having particular structure:

Theorem 3.3.1. *An optimal FKPP solution always exists, in which at most one item is fractionally selected.*

Proof. Let λ_i/w_i be the efficiency of item i, that is the relative profit for each packed unit. Assume to have a bin in which a set of items is fully packed, leaving a residual free space r; assume by contradiction that in an optimal solution it is convenient to complete the packing of the bin with two item fragments p and q, that is

$$\bar{x}_p^l > 0 \wedge \bar{x}_q^l > 0.$$

Their contribution to the objective value (3.3.6) is $\lambda_p \bar{x}_p^l + \lambda_q \bar{x}_q^l - (\mu_p + \mu_q + 2)$. Let us assume now, w.l.o.g., that item p is more efficient, that is $\frac{\lambda_p}{w_p} > \frac{\lambda_q}{w_q}$; in this case a better contribution to (3.3.1) would be obtained by using all the residual space r for the fragment of p: $\bar{x}_p^l = \frac{r}{w_p} \wedge \bar{x}_q^l = 0$ that would violate the initial assumption that fixing $\bar{x}_p^l > 0 \wedge \bar{x}_q^l > 0$ is optimal. The argument iteratively extends to solutions in which the number of fragments is higher than two. □

It immediately follows:

Corollary 3.3.2. *An optimal MP solution always exists, in which each selected column contains at most one fragmented item.*

Proof. Each MP column can always be found, by solving a pricing problem, as an optimal solution of a FKPP; a fragmented item is one fractionally selected in the FKPP. □

We remark that even if the columns of an optimal MP solutions have at most one fragmented item, a solution for the fm-BPPSPF may have more than one fragmented item per bin. For instance, any linear combination of two columns which are identical, except for the fragmented item, encodes a bin containing their common elements and both fragmented items.

To solve the FKPP we developed the following ad-hoc pseudo-polynomial time algorithm. It elaborates on the well-known dynamic programming approach for the $0-1$ KP [48]: let $M(\bar{I}, c)$ be the cost of an optimal FKPP solution in which only items in $\bar{I} \subseteq I$ are considered to be fully packed into the bin, and at most c units of capacity are consumed; the values $M(\bar{I}, c)$ can be recursively computed as follows:

$$M_{(\bar{I}\cup\{i\},c)} = \begin{cases} M_{(\bar{I},c)}, \text{ if } w_i > c \\ \max\{M_{(\bar{I},c)}; M_{(\bar{I},c-w_i)} + \lambda_i - \mu_i - 1\}, \text{ otherwise} \end{cases}$$

where $M_{\emptyset,c} = -\eta$ for each $c \leqslant C$. It is easy to keep track of the subset of items forming an optimal solution by storing which arguments yield maxima in the above expression.

According to Theorem 3.3.1, an optimal FKPP solution can be found by considering $|I| + 1$ independent cases:

- for each $i \in I$, i is the only fractionally selected item; in this case, the optimal solution value σ_i^* can be found by considering all possible ways of assigning $\hat{w}$ units of capacity to the fraction of i, and the residual space $C - \hat{w}$ to fully selected items, that is

$$\sigma_i^* = \max_{1 \leqslant \hat{w} \leqslant w_i} \{M_{(I\setminus\{i\},C-\hat{w})} + \hat{w} \cdot \frac{\lambda_i}{w_i} - \mu_i - 1.\}$$

- no item is fractionally selected; in this case the optimal solution value can be found as

$$\sigma^* = \max_{0 \leqslant \hat{w} \leqslant C} \{M_{(I,\hat{w})}\}.$$

The FKPP optimal solution value can finally be computed as

$$\max\left\{\sigma^*; \max_{i\in I} \sigma_i^*\right\}.$$

The complexity of the overall procedure is $O(C \cdot |I|^2)$. We also remark that the above procedure is suitable for multiple pricing, as each run produces a potentially different FKPP solution for each choice of the unique fractional item.

3.3.3 *Branch and bound*

Whenever a fractional optimal MP solution is found, whose value does not match that of a known integer fm-BPPSPF solution, we explore a search tree enforcing integrality through three branching rules. Let $\tilde{y}^l$ be the value of each y^l variable in a fractional MP solution for a certain node of the search tree.

In the first branching rule, we fix which items are fragmented and which are not. In particular, we search for an item $\hat{\imath}$ that is most fractionally selected to be fragmented in the MP solution, that is

$$\hat{\imath} \in \operatorname*{argmin}_{i \in I} \left\{ \left| \sum_{\gamma \in \Gamma} \bar{z}_i^\gamma \tilde{y}^\gamma - \frac{3}{2} \right| \right\}.$$

If such an item $\hat{\imath}$ has either $\sum_{\gamma \in \Gamma} \bar{z}_{\hat{\imath}}^\gamma \tilde{y}^\gamma = 1$ or $\sum_{\gamma \in \Gamma} \bar{z}_{\hat{\imath}}^\gamma \tilde{y}^\gamma = 2$, then the solution is integer with respect to the first branching rule, and we skip to the second one. Otherwise we create two children:

- in the first one we enforce $\hat{\imath}$ to be fragmented, by removing all RMP columns fully containing $\hat{\imath}$ and by generating no more columns of this kind; this is accomplished in our FKPP algorithm by removing item $\hat{\imath}$ from the set of available items I during the computation of the value σ^* and each value σ_i^* with $i \neq \hat{\imath}$;

- in the second one we enforce $\hat{\imath}$ not to be fragmented, by removing all RMP columns in which $\hat{\imath}$ is fragmented and by generating no more columns of this kind; that is, in our FKPP algorithm we skip the computation of $\sigma_{\hat{\imath}}^*$, and fix it in advance to $-\infty$.

We remark that, in both children, no substantial modification in the FKPP algorithm is required after branching.

Given a certain node of the search tree, let $\bar{J}$ be the set of items that were fixed to be fragmented by applying the first branching rule; in the second branching rule, we fix which pairs of items in $\bar{J}$ are assigned to the same bin and which are not. Indeed each fragmented item is assigned to more than one bin, and therefore it may be packed

together with more than one fragmented item. Let coefficient ψ_{ij}^{l} indicate whether two items i and j appear together in column l or not:

$$\psi_{ij}^{l} = \begin{cases} 1, \text{ if } \bar{z}_i^l + \bar{z}_j^l = 2 \\ 0, \text{ otherwise} \end{cases}$$

If $\sum_{l \in \Gamma} \psi_{ij}^{l} \cdot y^l$ is integer for each pair of fragmented items i and j in $\bar{J}$, then we move to the third branching rule. Otherwise, we select a pair of fragmented items

$$(\hat{\imath}, \hat{\jmath}) \in \underset{i \in \bar{J}, j \in \bar{J}}{\operatorname{argmin}} \left\{ \left| \sum_{l \in \Gamma} \psi_{ij}^{l} y^l - 0.5 \right| \right\}$$

and we create two children.

- In the first child we forbid $\hat{\imath}$ and $\hat{\jmath}$ to be assigned to the same bin. Besides removing from the RMP all columns not complying with this condition, such a constraint can be efficiently managed in the FKPP algorithm by removing item $\hat{\imath}$ from the set of available items I during the computation of the value $\sigma_{\hat{\jmath}}^{*}$ and vice versa. Therefore, in such columns the value of $\psi_{\hat{\imath}\hat{\jmath}}^{l}$ is always 0.

- In the second child we enforce $\hat{\imath}$ and $\hat{\jmath}$ to be packed together into at least one bin. It is still possible to handle such a condition without increasing the FKPP algorithm complexity as follows: let us suppose w.l.o.g. that one item has a better relative profit, that is $\lambda_{\hat{\imath}}/w_{\hat{\imath}} \geqslant \lambda_{\hat{\jmath}}/w_{\hat{\jmath}}$. We skip the computation of $\sigma_{\hat{\imath}}^{*}$ and $\sigma_{\hat{\jmath}}^{*}$ values, fixing them in advance to $-\infty$, and we compute instead a single $\sigma_{\hat{\imath}\hat{\jmath}}^{*}$ value, defined as

$$\begin{aligned} \sigma_{\hat{\imath}\hat{\jmath}}^{*} = \max_{1 \leqslant w \leqslant w_{\hat{\imath}} + w_{\hat{\jmath}} - 2} \{ & M_{(I \setminus \{\hat{\imath}, \hat{\jmath}\}, C - w)} \\ & - \mu_{\hat{\imath}} - \mu_{\hat{\jmath}} - 2 \\ & + \min\{w, w_{\hat{\imath}} - 1\} \cdot \frac{\lambda_{\hat{\imath}}}{w_{\hat{\imath}}} \\ & + (w - \min\{w, w_{\hat{\imath}} - 1\}) \cdot \frac{\lambda_{\hat{\jmath}}}{w_{\hat{\jmath}}} \}, \end{aligned}$$

  in which we consider all possible ways of packing $\hat{\imath}$ and $\hat{\jmath}$, forcing that at least one unit of the less profitable item $\hat{\jmath}$ is always packed, and therefore fixing $\psi_{\hat{\imath}\hat{\jmath}}^{l} = 1$.

We remark that, according to Theorem 3.1.1, each item can be fragmented in at most two bins, and therefore may be paired to at most two other fragmented items. Hence the number of $\sigma^*_{\hat{i}j}$ values is linear in the number of items, and their introduction does not affect the computational complexity of the FKPP algorithm.

Each pair of fragmented items packed together defines a bin in the integer solution. Therefore, by fixing which fragmented items are assigned to the same bin, we are implicitly and incrementally defining the chains in the BPG corresponding to a fm-BPPSPF integer solution. In each leaf of the second branching level the complete BPG is fixed.

In the third branching rule we assign unfragmented items to BPG chains. Let P be the set of chains in the BPG graph, and $\bar{J}_p$ be the set of fragmented items representing edges in chain $p \in P$. We define the following coefficient for each column $l \in \Gamma$, each item $i \in I \setminus \bar{J}$ and each chain $p \in P$:

$$\rho^l_{ip} = \begin{cases} 1 & \text{if } \exists j \in \bar{J}_p : z^l_i = 1 \wedge z^l_j = 1 \\ 0 & \text{otherwise;} \end{cases}$$

therefore, whenever

$$0 < \sum_{l \in \Gamma} \rho^l_{ip} y^l < 1$$

item i is fractionally assigned to multiple chains. If none of these values is fractional, then we stop branching and proceed to a final feasibility check, as described in Subsection 3.3.4. Otherwise we select the pair $(\hat{i}, \hat{p})$ having highest $\sum_{l \in \Gamma} \rho^l_{ip} y^l$ value, among those being fractional, and we create two children:

- in the first child we forbid $\hat{i}$ to be assigned to chain $\hat{p}$, by removing from the RMP all columns in which i is assigned to a bin whose fragmented items are in p, and in the FKPP algorithm by removing item $\hat{i}$ from the set of available items during the computation of each σ^*_j and $\sigma^*_{j'j''}$ value having either j, j' or j'' in $\bar{J}_p$;
- in the second child we impose $\hat{i}$ to be assigned to chain $\hat{p}$; this is done by equivalently forbidding to assign $\hat{i}$ to any chain $p \neq \hat{p}$ and by excluding i from the computation of value σ^*.

Still, the computational complexity of the FKPP pricing algorithm does not increase after branching.

The branching tree is explored with a best-bound-first order. We perform branching when column generation is over, using the first applicable branching rule. This policy gave best results in a set of preliminary experiments.

If no fractionality can be detected in a particular node of the search tree, triggering one of the branching rules described in the previous section, then the corresponding partial solution might either yield a feasible final fm-BPPSPF solution or not. In fact, through our branching strategy we impose the integrality constraints for what concerns the assignment of items to chains containing fragmented items. However, the assignment of items to bins with no fragmented items may still violate integrality constraints. Therefore we perform a final feasibility check: if such a test succeeds, then the corresponding fm-BPPSPF solution is also improving with respect to the current incumbent. Otherwise, the partial solution is discarded by skipping the search tree node.

3.3.4 *Feasibility check*

Let us suppose that a partial fm-BPPSPF solution is found through branching. It includes a BPG composed by a set P of chains, each defined by a set J_p of fragmented items, together with a partition $I_0, I_1, \ldots, I_{|P|}$ of I, where I_0 represents the set of items to be packed into bins containing no fragmented items, and each subsequent I_p represent the set of items to be packed into the set of bins belonging to the same chain $p \in P$. Then, the feasibility of such a fm-BPPSPF partial solution can be computed as follows.

(a) for each $p \in P$, apply NF_f to I_p; let $len(p)$ be the number of bins used by NF_f on I_p: if $len(p) > |J_p| + 1$, then the partial fm-BPPSPF solution cannot be completed in any feasible way;

(b) otherwise, build a traditional BPP instance containing all items in I_0 and having $|B| - \sum_{p \in P} len(p)$ available bins of capacity C.

(c) solve a BPP feasibility problem on such an instance: if a BPP infeasibility is detected, then the partial fm-BPPSPF solution cannot be completed in any feasible way.

In fact, in step (a) NF_f always produces on a single chain primitive solutions using the minimum number of bins, and therefore requiring the minimum number of fragmented items: if it is larger than the

number of fragmented items defining the chain, then no feasible accommodation of the items into the chain is possible. At the same time, in step (c), no fragmentation is allowed on the items in I_0: since they cannot belong to the chains, each of them must be inserted into one of the remaining bins with no fragmentations; thus, the subproblem turns out to be a traditional BPP.

3.4 IMPROVEMENT TECHNIQUES

In order to enhance the performances of our algorithms, in Subsection 3.4.1 we introduce problem reduction techniques, to be applied during branching, focusing also on symmetry reduction. In Subsection 3.4.2 we describe upper bounding heuristics that, being able to quickly provide good fm-BPPSPF solutions, help to perform early pruning. Both techniques proved to enhance significantly the overall performance of the method.

3.4.1 *Reduction*

VARIABLE FIXING. Exploiting Theorem 3.1.1, as soon as $|B| - 1$ items are forced to be fragmented by the first branching rule, we mark the remaining ones as unfragmented, restricting our search to primitive solutions only. Similarly, still according to Theorem 3.1.1, as soon as a fragment of an item i is forced to appear together with two different item fragments by the second branching rule, to keep only primitive solutions we forbid any other item fragment to appear with i.

CHECKING ON CAPACITY CONSTRAINTS. After branching by the second or third rule, we perform the following test based on the solution of Subset-Sum Problems (SSP) [55]. First, suppose a BPG chain p is given, corresponding to a set B_p of bins, together with the size of each fragmented item to be assigned to each bin; let C_j be the residual capacity of each bin $j \in B_p$ after item fragments are accommodated. Furthermore, let $\bar{J}$ be the set of items fixed to be fragmented through branching.

Observation 3.4.1. *The problem of maximizing the overall usage of capacity of bins in* B_p *is a Multiple Knapsack Problem (MKP) [48].*

In fact, such a problem can be stated as:

$$\max \quad \sum_{\substack{i \in I \setminus \bar{J} \\ j \in B_p}} w_i \cdot x_{ij} \tag{3.4.1}$$

$$\text{s.t.} \quad \sum_{i \in I \setminus \bar{J}} w_i \cdot x_{ij} \leqslant C_j \qquad \forall j \in B_p \tag{3.4.2}$$

$$\sum_{j \in B_p} x_{ij} \leqslant 1 \qquad \forall i \in I \setminus \bar{J} \tag{3.4.3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I \setminus \bar{J}, \forall j \in B_p \tag{3.4.4}$$

where $x_{ij} = 1$ is item i is assigned to bin j, 0 otherwise.

Observation 3.4.2. *Let S^*_{MK} be an optimal solution value of problem (3.4.1) – (3.4.4). If the capacity on remaining bins is less than the weight of unselected items, that is*

$$s \sum_{i \in I \setminus J} w_i - S^*_{MK} > |B \setminus B_p| \cdot C \tag{3.4.5}$$

then no feasible fm-BPPSPF solution can be obtained by completing the partial one.

Such an observation cannot be straightly applied in our framework since (a) the size of each fragment is not fixed by branching and (b) solving MKPs is computationally too demanding. Therefore, we restrict to the following sufficient condition.

Theorem 3.4.1. *Let P be the set of chains in the BPG of a partial solution, in which each chain p corresponds to a set B_p of bins; let $\bar{I}_p$ be the set of items forced to be assigned to bins in B_p without fragmentations and let $\bar{J}_p$ the set of fragmented items defining chain p. For a particular $p \in P$, let S^*_{SSP} be the optimal solution value of a SSP in which a single bin of capacity $|B_p| \cdot C - \sum_{i \in \bar{I}_p \cup \bar{J}_p} w_i$ must be filled with items in $I \setminus \bigcup_{p \in P} \bar{I}_p \cup \bar{J}_p$. Then if the capacity of bins that do not belong to chain p is less than the weight of unselected items, that is*

$$\sum_{i \in I \setminus \bar{I}_p \cup \bar{J}_p} w_i - S^*_{ss} > |B \setminus B_p| \cdot C \tag{3.4.6}$$

then no feasible fm-BPPSPF solution can be obtained by completing the partial one.

Proof. The result follows directly from observations 3.4.1 and 3.4.2, and by observing that S^*_{SSP} can be obtained by surrogating constraints

(3.4.2) in (3.4.1) – (3.4.4), and is therefore an upper bound on S^*_{MK}: condition (3.4.6) implies condition (3.4.5). □

The result holds, as a special case, if a bin is fixed during branching to contain no fragmented items. This situation occurs when less than $|B|$ items are fixed to be fragmented, and the remaining ones are fixed not to be fragmented through branching; in this case each singleton in the BPG represents a bin containing no fragmented items.

Therefore, after applying the second or third branching rule, we run the feasibility check algorithm described in Pseudocode 3.4.1, where we assume $SS(I, w, C)$ to be a function returning an optimal solution value for a SSP instance involving a single bin of capacity C, a set I of items and a weight vector w.

function ISNODEFEASIBLE(I, B, w, C, P)
  for all $p \in P$ **do**
    $C_p \leftarrow (|B_p| - 1) \cdot (C - \sum_{i \in \bar{I}_p \cup \bar{J}_p} w_i)$
    $I_{SS} \leftarrow SS(I \setminus \bigcup_{p \in P} \bar{I}_p \cup \bar{J}_p, w, C_p)$
    if $\sum_{i \notin I_{SS} \cup \bar{I}_p \cup \bar{J}_p} w_i > (|B| - |B_p|) \cdot C$ **then**
      return False
    end if
  end for
  return True
end function

Pseudocode 3.4.1: Algorithm that verifies the feasibility of a node

After applying the third branching rule, before running algorithm 3.4.1, in order to avoid useless SSP computations we perform the following quick preliminary check:

Remark 3.4.1. *Let p be a BPG chain in a partial solution, and let $\tilde{I}_p$ be the set of items whose assignment to bins of chain p is forbidden by branching decisions. If the sum of the weights of the items in $\tilde{I}_p$ is more than the residual space, that is*

$$\sum_{i \in \tilde{I}_p} w_i > (|B| - |B_p|) \cdot C \tag{3.4.7}$$

then no feasible fm-BPPSPF solution can be obtained by completing such a partial solution.

If either test fails, the node is pruned.

SYMMETRY REDUCTION. To reduce symmetries, we explore only branching nodes with particular features. Let p be a chain in a partial

BPG, obtained in first and second branching levels, and let $\underline{w}$ be the minimum weight of a fragmented item forming p: we never generate columns in which an item i having $w_i > \underline{w}$ is assigned to one of the bins of p. In fact, according to Theorem 3.1.3, an optimal fm-BPPSPF solution always exists, in which the fragmented items are the largest ones in each chain. We enforce this condition directly in the FKPP algorithm, by excluding from the set I during the computation of each σ_i^* and σ_{ij}^* value, each item having weight larger than that of a fragmented item in the corresponding partial BPG chain. Beside reducing symmetries, this removal helps to reduce the computational effort in solving the FKPP subproblems.

3.4.2 *Heuristics*

In order to obtain a good upper bound to the problem, we experimented with different primal heuristics, including two drawn from the literature.

GREEDY ALGORITHMS. We first considered the *Next-Fit item fragmentation (NF_f)* proposed in [56], reported also in Pseudocode 3.1.1. Then, we experimented also a *Best-Fit-Decreasing item fragmentation (BFD_f)* method [56]: we sort the items in non-increasing weight order, and we pack them without fragmentations using a Best-Fit BPP heuristic, until the sum of residual capacity in the bins is at least equal to the sum of the weights of unpacked items; then we complete the solution by exploiting the residual capacity with a NF_f policy.

SUBSET-SUM HEURISTIC (SSH). Then we designed an algorithm that is based on the iterative resolution of SSPs. The idea behind this approach is the following: items that are packed with fragmentations into a BPG chain p of $|B_p| - 1$ bins, can also be packed without fragmentations into a single bin of capacity $|B_p| \cdot C$.

Let us assume, as in the previous Subsection, to have a procedure $SS(I, w, C)$ receiving as input a SSP instance, that is a set I of items, a vector w of weights and a capacity C, and giving as output a set I_{SS}, corresponding to an optimal SSP solution. Our SSP Heuristic (SSH) iteratively tries to create short chains first: the shorter are the chains in a fm-BPPSPF solution, the better is the solution value, but the higher is the chance of leaving unused capacity in the bins of the chain. If an infeasibility is detected, by comparing the weight of unassigned items

with the capacity of unused bins, then the tentative chain length is increased. A formal description is reported in Pseudocode 3.4.2.

function SSH(I, B, w, C)
  $P \leftarrow \emptyset$
  $n \leftarrow |B|$
  $m \leftarrow 1$
  while $I \neq \emptyset$ **do**
    $I_{SS} \leftarrow SS(I, w, C \cdot m)$
    if $\sum_{i \in I \setminus I_{SS}} w_i > (n - m) \cdot C$ **then**
      $m \leftarrow m + 1$
    else
      use NF_f on I_{SS} to produce a chain p
      $P \leftarrow P \cup \{p\}$
      $n \leftarrow n - m$
      $I \leftarrow I \setminus I_{SS}$
    end if
  end while
  return P
end function

Pseudocode 3.4.2: SSH algorithm

ROUNDING. We designed also a rounding procedure, that is executed at the end of the column generation process of each node of the search tree. We first fix as fragmented those items i having

$$\sum_{\gamma \in \Gamma} \bar{z}_i^\gamma \cdot \tilde{y}^\gamma > 3/2,$$

and form chains from pairs of fragmented items (i, j) having

$$\sum_{l \in \Gamma} \psi_{ij}^l \cdot y^l > 1/2;$$

in this way the BPG, and therefore the value of the corresponding fm-BPPSPF solution, are completely defined. Then, if such a BPG could improve the incumbent, two different heuristic feasibility checks are performed, trying to fit into the chains all unfragmented, and therefore unassigned, items. First, each unassigned item is inserted into a chain where its assignment is maximum in the fractional solution: if capacity constraints can be respected in this way, then an improving incumbent is found. Otherwise, we consider the BPG chains in an arbitrary order: we solve for each of them a SSP on the set of unassigned items, and remove from the set of unassigned items those selected in an optimal SSP solution. If after considering all the chains some items

are still unassigned, then the rounding procedure fails in finding any improving solution.

After preliminary experiments, we found it useful to run the greedy and SSH algorithms once, before solving the root node relaxation, and to run the rounding heuristic once at each node of the branching tree, when column generation is over. Heuristics NF_f and SSH were used also to initialize the RMP, while BF_f was not, being unable to guarantee that primitive solutions are generated.

3.5 COMPUTATIONAL RESULTS

We implemented our algorithms in C++, using the framework SCIP [1] version 2.1.1 keeping the default options.

The simplex algorithm implemented in CPLEX 12.4 is used to solve the LP subproblems: the solver automatically switches between primal and dual methods. At each column generation iteration we include columns with negative reduced cost only, corresponding (a) to value σ^* (b) to the minimum σ_i^* value, and (c) to the minimum σ_{ij}^* value. These encode, at each iteration, the best columns containing zero, one or two fragmented items, respectively. This policy produced better results than more aggressive multiple pricing methods. The SSPs are solved by a C++ implementation of a dynamic programming algorithm similar to the one described in section 3.3, while the feasibility BPPs are solved using the Constraint Programming Optimizer of CPLEX 12.4.

Since there are no specific fm-BPPSPF instances described in the literature, we produced a randomly generated benchmark, considering different instance sizes, distribution of the weights and amount of residual capacity in the bins.

In particular, we considered instances including 10, 15 or 20 items as, surprisingly, these problems are already out of reach for a state-of-the-art general purpose solver. Then we considered three types of weight distribution:

LARGE : weights are uniformly drawn to be integers between 50% and 90% of the capacity of the bins;

SMALL : weights are uniformly drawn to be integers between 10% and 50% of the capacity of the bins;

FREE : weights are uniformly drawn to be integers between 10% and 90% of the capacity of the bins;

Finally, with respect to residual capacity, we consider two types of instances:

TIGHT : no residual space, that is $W = \sum_{i \in I} w_i = C \cdot |B|$

LOOSE : 10% of residual space, that is $W = \sum_{i \in I} w_i = 0.9 \cdot C \cdot |B|$

In order, to generate such instances we first fixed the number of items, the number of bins and their capacity. Let $\underline{w}$ and $\bar{w}$ be respectively the weights lower bound and upper bound. We created a set R containing values 0, $W - \underline{w} \cdot |I|$ and $|I| - 1$ additional values randomly drawn from a uniform distribution in the range $[0, W - \underline{w} \cdot |I|]$. We sorted its elements in non decreasing order. For each pair of elements r_i, r_{i+1} sequentially chosen from R, we set the weight of item i to be $w_i = r_{i+1} - r_i + \underline{w}$. This procedure was repeated until no item i was found to have $w_i > \bar{w}$.

We created an instance class for each combination of instance size, weight distribution and residual capacity, and generated ten instances for each class, obtaining a dataset of 180 instances. We also performed preliminary tests on instances having $W = \sum_{i \in I} w_i = 0.8 \cdot C \cdot |B|$ or less, finding them easier to be solved. Indeed, when capacities are not tight, the fragmentation models are of no particular interest, as they basically behave as traditional BPPs.

As a benchmark we considered the state-of-the-art solver IBM ILOG CPLEX [36] version 12.4, using the mathematical programming model described in Section 3.2, and keeping default settings.

All the tests have been performed on a PC equipped with an Intel(R) Core2 Duo CPU E6850 at 3.00GHz and 4GB of memory.

3.5.1 *Root lower bound*

In a first round of experiments we compared the performances for obtaining a lower bound of both our Column Generation algorithm (CG) and CPLEX (CPX). In Table 1 we report, for each instance class and for both methods, the time spent at the root node, and the average gap $(\mathcal{F}^* - LB)/\mathcal{F}^*$ between the corresponding lower bound LB and the best known fm-BPPSPF solution $\mathcal{F}^*$. The results show that the execution time of both methods is very similar. Instead, in terms of bound quality CG is better than CPX in all but three classes; in these three classes the average gap is the same, and in two of them both methods are able to fully close the gap at the root node.

Class			CPX		CG	
|I|	w.	cap.	t (s)	Gap (%)	t (s)	Gap (%)
10	big	tight	0.08	69.67	0.08	**23.33**
10	big	loose	0.03	100.00	0.06	**53.33**
10	free	tight	0.02	100.00	0.06	**69.17**
10	free	loose	0.02	20.00	0.04	**0.00**
10	small	tight	0.01	100.00	0.07	**35.00**
10	small	loose	0.00	**0.00**	0.02	**0.00**
15	big	tight	0.20	56.11	0.10	**19.31**
15	big	loose	0.06	100.00	0.09	**45.00**
15	free	tight	0.04	100.00	0.11	**70.00**
15	free	loose	0.04	10.00	0.05	**0.00**
15	small	tight	0.05	100.00	0.13	**55.00**
15	small	loose	0.02	**0.00**	0.03	**0.00**
20	big	tight	0.52	69.55	0.13	**19.84**
20	big	loose	0.12	100.00	0.14	**50.00**
20	free	tight	0.07	100.00	0.23	**76.00**
20	free	loose	0.06	**0.00**	0.10	**0.00**
20	small	tight	0.07	100.00	0.27	**50.00**
20	small	loose	0.03	**0.00**	0.06	**0.00**

Table 1: Lower bound at the root node

3.5.2 *Root upper bound*

In the second round of experiments we compared the quality of the primal bound obtained at the root node by both CPLEX (CPX), our Column Generation algorithm (CG), and each of the heuristics described in Section 3.4.2.

In Table 2 we report for each instance class and for each method, the average gap $(UB - \mathcal{F}^*)/(UB)$ between the corresponding best primal bound UB and the best known fm-BPPSPF solution $\mathcal{F}^*$. The UB value is computed picking the best solution between the results given by NF_f, BFD_f, SSH and the rounding heuristic executed after computing the lower bound of the root node. For each instance class we report also the maximum gap given by heuristics NF_f, BFD_f and SSH. Furthermore, for CPX and CG we report also the computing time: that of CG includes the running time of all heuristics.

The results show that, while execution time is very similar, the primal bound obtained with our heuristics is always better than CPX. In fact, they obtain the fm-BPPSPF best known solution in more that 90% of the instances.

SSH outperforms other heuristics, except for classes with loose capacity and free weights, where BFD_f seems to perform better.

3.5.3 *Solving time*

In the last round of experiments we compared the performances of CPLEX (CPX) and our full Branch and Price algorithm (BP) in solving fm-BPPSPF instances to proven optimality. A time limit of two hours was given to each run. Our results are reported in Table 3.

It consists of four blocks. The first one indicates the instance class details. The second and third ones report the performances of CPX and BP, respectively. The last one contains details about the BP solution process. In particular, the second and third blocks are composed by three columns each, reporting in turn the number of instances solved to proven optimality within the time limit, the average computing time on them and the average duality gap on the remaining ones; the latter is computed as $(UB - LB)/UB$, where UB and LB are the best lower and upper bounds found by the algorithm within the time limit. In the last block we report the average number of times BP runs each branching rule for each class of instances.

Class			CPLEX		CG		Average Gap			Maximum Gap		
							$\mathbf{NF_f}$	$\mathbf{BFD_f}$	**SSH**	$\mathbf{NF_f}$	$\mathbf{BFD_f}$	**SSH**
|I|	w.	cap.	t (s)	Gap (%)	t (s)	Gap (%)	Gap (%)	Gap (%)	Gap (%)	Gap (%)	Gap (%)	Gap (%)
10	big	tight	0.08	11.67	0.08	**0.00**	11.67	11.67	**0.00**	33.33	33.33	**0.00**
10	big	loose	0.03	2.50	0.06	**0.00**	50.00	40.00	**0.00**	50.00	40.00	**0.00**
10	free	tight	0.02	10.00	0.06	**0.00**	17.50	17.50	**0.00**	25.00	25.00	**0.00**
10	free	loose	0.02	80.00	0.04	**0.00**	95.00	**6.67**	30.00	100.00	**66.67**	100.00
10	small	tight	0.01	10.00	0.07	**0.00**	15.00	15.00	**0.00**	50.00	50.00	**0.00**
10	small	loose	0.00	30.00	0.02	**0.00**	100.00	**0.00**	**0.00**	100.00	**0.00**	**0.00**
15	big	tight	0.20	10.33	0.10	**0.00**	13.00	13.00	**0.00**	20.00	20.00	**0.00**
15	big	loose	0.06	2.00	0.09	**0.00**	55.56	48.57	**0.00**	55.56	50.00	**0.00**
15	free	tight	0.04	21.67	0.11	**1.67**	22.86	22.86	**1.67**	28.57	28.57	**16.67**
15	free	loose	0.04	95.00	0.05	**0.00**	98.57	**7.50**	20.00	100.00	**75.00**	100.00
15	small	tight	0.05	38.33	0.13	**0.00**	42.50	42.50	**0.00**	50.00	50.00	**0.00**
15	small	loose	0.02	60.00	0.03	**0.00**	100.00	**0.00**	**0.00**	100.00	**0.00**	**0.00**
20	big	tight	0.52	13.27	0.13	**0.00**	23.08	22.44	**0.00**	30.77	30.77	**0.00**
20	big	loose	0.12	4.29	0.14	**0.00**	50.00	40.00	**0.00**	50.00	40.00	**0.00**
20	free	tight	0.07	28.19	0.23	**1.67**	35.56	35.56	**1.67**	44.44	44.44	**16.67**
20	free	loose	0.06	100.00	0.10	**0.00**	100.00	**0.00**	70.00	100.00	**0.00**	100.00
20	small	tight	0.07	55.00	0.27	**10.00**	60.00	59.00	**10.00**	60.00	60.00	**33.33**
20	small	loose	0.03	30.00	0.06	**0.00**	100.00	**0.00**	**0.00**	100.00	**0.00**	**0.00**

Table 2: Upper bound at the root node

We first observe that item weights and capacity type affect the performances of both methods more than the instance size: the larger the items are, and the tighter the capacity is, the harder it is to solve an instance. The experiment shows also that, for CPX, instances having $|I| = 15$ are on the edge between solvable and unsolvable in two hours, and those having $|I| = 20$ are out of reach, except for classes with free or small weights and loose capacity. Our algorithm solves about 39% more instances and shows a much more stable behavior; furthermore, on the unsolved instances, the duality gap left by BP is always smaller. A similar observation can be made for computing times: BP is in general much faster than CPX, disregarding instance class.

In a preliminary round of experiments we also evaluated the performance of CPX with different inequalities derived from the definition of primitive packing, like

$$\sum_{j \in B} z_{ij} \leqslant 2, \forall i \in I,$$

without obtaining substantial improvements. Such behaviour is due to the weakness of the original formulation, which never provide a tight bound. In fact CPX finds easily a good solution to the problem, and most of its computing time is spent exploring the search tree to prove the optimality.

In terms of number of nodes generated during the computation, BP spends most of the time in fixing the pairs of fragmented items to be packed together, except for classes with big items and loose capacity, where it is harder to fix which items are to be fragmented.

3.6 CONCLUSIONS

In this chapter we tackled a variant of the Bin Packing Problem in which each item can be fractionally assigned to different bins at a price. We performed a theoretical investigation, designed both exact and heuristic methods and performed an experimental campaign.

As a main theoretical result, we could prove that optimal solutions exist, having very particular structure, and being representative of a combinatorial number of equivalent solutions. By using these theoretical properties and Dantzig-Wolfe decomposition, we were able to design both a mathematical programming algorithm, and fast and accurate heuristics.

Class			CPLEX			BP					
						Results			Branching		
|I|	w.	cap.	S	t (s)	Gap (%)	S	t (s)	Gap (%)	1^{st}	2^{nd}	3^{rd}
10	big	tight	6	427.18	8.33	10	137.39	0	27	807	91
10	big	loose	10	140.4	0	10	0.25	0	23	0	0
10	free	tight	10	36.33	0	10	0.4	0	11	14	4
10	free	loose	10	0.03	0	10	0.03	0	0	0	0
10	small	tight	10	0.76	0	10	0.09	0	1	0	0
10	small	loose	10	0.01	0	10	0.02	0	0	0	0
15	big	tight	0	-	17.36	1	0.33	17.08	151	5364	291
15	big	loose	0	-	65	10	5.02	0	180	0	0
15	free	tight	0	-	56.67	6	5.97	6.67	92	1515	199
15	free	loose	10	0.09	0	10	0.05	0	0	0	0
15	small	tight	7	1090.17	16.67	10	0.5	0	7	1	4
15	small	loose	10	0.02	0	10	0.03	0	0	0	0
20	big	tight	0	-	29.88	0	-	18.84	462	2793	209
20	big	loose	0	-	93.33	3	1822.92	11.67	5406	1	0
20	free	tight	0	-	82.29	3	124.65	11.67	347	966	980
20	free	loose	10	0.1	0	10	0.1	0	0	0	0
20	small	tight	0	-	86.67	10	2.25	0	8	2	11
20	small	loose	10	0.03	0	10	0.05	0	0	0	0

Table 3: Results obtained within a time limit of one hour per instance

Our experimental study revealed that a state-of-the-art general purpose solver like CPLEX using a compact formulation fails in optimizing even instances of very limited size, being outperformed by our algorithms in terms of both solutions quality and computing time.

One of our heuristics showed to be far more effective than those proposed in the literature, reaching global optima in about 90% of the instances, and requiring negligible computing time.

Through our experiments, we also found out that the distribution of item weights and tightness of capacity constraints seem to contribute more than the number of items in making an instance challenging.

4

INTRODUCING A CHAIN BASED FORMULATION

We now discuss how to improve the previous algorithm by further exploiting properties on the structure of the solutions. We devise a new formulation that (a) is more flexible, as can be applied to different variants of BPPIF described above, including size increasing variants and (b) is more effective, solving instances of one order of magnitude larger in orders of magnitude faster.

For the ease of exposition, in Section 4.1 we present useful theoretical properties on the structure of optimal solutions that lead to a pure combinatorial new mathematical programming model of the problem. We then present its extended formulation obtained by applying Dantzig-Wolfe decomposition method [30]. In Section 4.2 we detail our new exact algorithm to solve it. Then, in Section 4.3 we discuss on how to extend it in order to tackle the size increasing variant. In Section 4.4 we present our experimental analysis. Finally, in Section 4.5 we summarize our results and collect some brief conclusions.

4.1 PROPERTIES EXTENSION AND REFORMULATION

Following the definition of primitive solution in Subsection 3.1.1, it is easy to prove that such a structure directly influences the cost of a packing by representing a primitive solution through the construction of a *Bin Packing Graph (BPG)*.

It is easy to observe:

Theorem 4.1.1. *The cost of a primitive solution is the sum of the length of all the chains in its BPG minus the number of chains.*

Proof. In fact, let l_k be the length of a chain k in the BPG. The cost of a single chain is the number of its edges, that is $l_k - 1$. The overall number of fragmentation is the sum of all edges of a BPG, that is

$$\sum_{k \in BPG} (l_k - 1) = \sum_{k \in BPG} l_k - \sum_{k \in BPG} 1.$$

□

Furthermore, because Theorem 3.1.3, the following holds:

Proposition 4.1.1. *Let a partitioning of the set of items be given, in which each class corresponds to the subset of items packed into bins of the same chain. Then, a feasible primitive solution can be obtained using the* Next-Fit with Item Fragmentation (NF_f) *on each class independently.*

This method does not affect the cost of the solution, since the length of each chain is fixed and thus the overall number of fragmentations does not change.

By exploiting these properties, we can model the fm-BPPSPF as the problem of optimally packing items into chains instead of single bins (Figure 6).

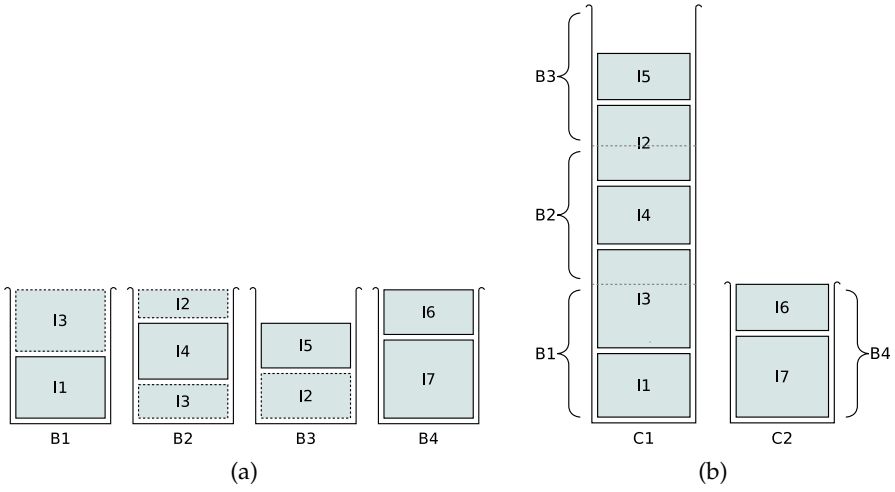

Figure 6: A primitive solution (a) with its corresponding chain representation (b).

Let K be the index set of the chains. Let l_k be a variable representing the length of each $k \in K$; that is, each $k \in K$ includes a set of l_k bins, involves $l_k - 1$ item splits and provides an overall capacity of $l_k \cdot C$. Model (3.2.1)–(3.2.6) can be reformulated as follows:

$$\min \quad \sum_{k \in K} l_k - v_k \tag{4.1.1}$$

$$\text{s.t.} \quad \sum_{k \in K} z_{ik} = 1 \qquad \forall i \in I \tag{4.1.2}$$

$$\sum_{k \in K} l_k \leqslant |B| \tag{4.1.3}$$

$$\sum_{i \in I} w_i \cdot z_{ik} \leqslant C \cdot l_k \qquad \forall k \in K \tag{4.1.4}$$

$$v_k \leqslant l_k \qquad \forall k \in K \tag{4.1.5}$$

$$z_{ik} \in \mathbb{B} \qquad \begin{matrix} \forall i \in I \\ \forall k \in K \end{matrix} \tag{4.1.6}$$

$$l_k \in \mathbb{N} \qquad \forall k \in K \tag{4.1.7}$$

$$v_k \in \mathbb{B} \qquad \forall k \in K \tag{4.1.8}$$

where each binary variable z_{ik} is set to 1 if item i is packed into chain k, and each binary variable v_k is set to 1 if chain k contains at least one item.

The objective function (4.1.1) minimizes the number of fragmentations. Each item can be split among bins in the same chain, but no fractional assignment of items to different chains is allowed: constraints (4.1.2) impose that each item is fully packed in a single chain. Constraints (4.1.4) ensure that the capacity of each chain is not exceeded, and constraint (4.1.3) guarantees that at most $|B|$ bins are used. Constraints (4.1.5) enforce consistency between variables, so that a chain is used only if its length is at least one.

We remark that a solution of (4.1.1) – (4.1.8) encodes no information on which items are split, nor on which items are packed into the same bin of each chain. In fact, due to Proposition 4.1.1, it is possible to obtain a feasible solution of fm-BPPSPF starting from any feasible solution of (4.1.1) – (4.1.8) with post processing, by applying the NF_f algorithm on each chain independently.

4.1.1 *Extended formulation*

From a continuous relaxation point of view, neither formulation (3.2.1) – (3.2.6) nor (4.1.1) – (4.1.8) offer a significant lower bound: in the first model, it is possible to iteratively pack all the items and then fix $z_i = x_i$ for each item. This provide a feasible solution for the continuous relaxation with objective function equal to 0. Likewise in the chain

model it is possible to fix $l_k = v_k = 1$ for each chain k, and then pack all the items assigning a fractional value to z_{ik}. Unfortunately also additional contraints $z_{ik} \leqslant v_k$ would not improve the continuous relaxation, because variables v_k are always greater than variables z_{ik}.

Therefore, we propose a reformulation obtained through Dantzig-Wolfe decomposition [30]. Let

$$z_k = (z_{1k}, z_{2k}, \ldots, z_{|I|k})$$

and

$$w = (w_1, w_2, \ldots, w_{|I|}).$$

Let, for each k in K,

$$\Omega_k = \left\{ (z_k, v_k, l_k) \in \mathbb{B}^{|I|} \times \mathbb{B} \times \mathbb{N} \mid w^T \cdot z_k \leqslant C \cdot l_k \wedge v_k \leqslant l_k \right\}$$

be the set of feasible integer points with respect to constraints (4.1.4) – (4.1.8). We relax integrality conditions, but replace each Ω_k with the convex hull of its P_k extreme integer points

$$\Gamma_k = \left\{ (\bar{z}_k^1, \bar{v}_k^1, \bar{l}_k^1), \ldots, (\bar{z}_k^{P_k}, \bar{v}_k^{P_k}, \bar{l}_k^{P_k}) \right\}$$

and then we impose

$$(z_k, u_k, l_k) = \sum_{p \in P_k} (\bar{z}_k^p, \bar{v}_k^p, \bar{l}_k^p) \cdot y_k^p \tag{4.1.9}$$

with $y_k^p \geqslant 0$ for each $k \in K$, $p \in \Gamma_k$, and $\sum_{p \in \Gamma_k} y_k^p = 1$ for each $k \in K$. That is, each point is represented as a linear convex combination of points in Γ_k, and variables y represent coefficients in such a combination.

The model obtained by replacing in the continuous relaxations of model (4.1.1) – (4.1.8) the vectors (z_k, u_k, l_k) as indicated in (4.1.9), and by making explicit the vector indices via Γ_k is

$$\min \quad \sum_{k\in K}\sum_{p\in\Gamma_k}(\bar{l}_k^p-\bar{v}_k^p)\cdot y_k^p \tag{4.1.10}$$

$$\text{s.t.} \quad \sum_{k\in K}\sum_{p\in\Gamma_k}\bar{z}_{ik}^p\cdot y_k^p=1 \qquad \forall i\in I \tag{4.1.11}$$

$$\sum_{k\in K}\sum_{p\in\Gamma_k}\bar{l}_k^p\cdot y_k^p\leqslant |B| \tag{4.1.12}$$

$$\sum_{p\in\Gamma_k}y_k^p=1 \qquad \forall k\in K \tag{4.1.13}$$

$$y_k^p\geqslant 0 \qquad \begin{matrix}\forall k\in K\\ \forall p\in\Gamma_k\end{matrix} \tag{4.1.14}$$

Constraints (4.1.11) can be relaxed in $\geqslant$ form, as an optimal solution always exists in which no item is assigned to bins more than once. Constraints (4.1.13) can be relaxed in $\leqslant$ form by observing that an empty pattern with $\bar{l}_k^p=0$, $\bar{v}_k^p=0$ and $\bar{z}_{ik}^p=0$ always exists for each $k\in K$; in fact selecting such a pattern is equivalent to setting all the corresponding y_k^p variables to 0. From this relaxation we also observe that objective function (4.1.10) can be rewritten as

$$\sum_{k\in K}\sum_{p\in\Gamma_k}(\bar{l}_k^p-1)\cdot y_k^p$$

since any chain that is not empty must have $\bar{v}_k^p=1$.

We also observe that since bins are identical, so are the sets Γ_k. Therefore, we consider a single representative $\Gamma=\bigcup_{k\in K}\Gamma_k$, and aggregate constraints (4.1.13) as

$$\sum_{p\in\Gamma}y^p\leqslant |K|. \tag{4.1.15}$$

Furthermore, constraints (4.1.15) can be removed from the model since it is redundant due to constraint (4.1.12). After a rewriting in canonical form, we obtain the following Master Problem (MP):

$$\min \quad \sum_{p\in\Gamma}(\bar{l}^p-1)\cdot y^p \tag{4.1.16}$$

$$\text{s.t.} \quad \sum_{p\in\Gamma}\bar{z}_i^p\cdot y^p\geqslant 1 \qquad \forall i\in I \tag{4.1.17}$$

$$-\sum_{p\in\Gamma}\bar{l}^p\cdot y^p\geqslant -|B| \tag{4.1.18}$$

$$y^p\geqslant 0 \qquad \forall p\in\Gamma \tag{4.1.19}$$

Observation 4.1.1. *The lower bound provided by the MP is at least as tight as that given by the continuous relaxation of model (4.1.1) – (4.1.8).*

The observation directly follows from the Dantzig-Wolfe decomposition principle. We further observe that, although the continuous relaxations of models (3.2.1) – (3.2.6) and (4.1.1) – (4.1.8) are equivalent, their Dantzig-Wolfe decompositions are not. In particular:

Proposition 4.1.2. *The lower bound provided by the MP is at least as tight as that given by the extended formulation (3.2.12) – (3.2.16).*

In fact the latter consists of an extended formulation with one column for each feasible assignment pattern of items to bins. Intuitively, given an optimal MP solution $\tilde{y}^p$, we can run NF_f on each pattern $p \in \Gamma$ of length l_p, build l_p assignment patterns of items to bins, corresponding to the l_p bins in the NF_f solution, and select each of them for a value $\tilde{y}^p$. This solution is feasible for the extended formulation (3.2.12) – (3.2.16). Indeed, we experimentally observed it to be often suboptimal, thereby providing weaker bounds.

4.2 ALGORITHMS

Our framework remains the same as the one described in Section 3.2: due to the exponential number of columns in the set Γ, we recur to column generation techniques to solve the MP. We initialize the RMP with a small subset of columns as described in Subsection 4.2.1, and solve the RMP to optimality. We search for variables having negative reduced cost by solving a particular variant of the 0-1 Knapsack Problem (KP) (see Subsection 4.2.2). We add such variables to the RMP and repeat the column generation process until no negative reduce cost variable is found, meaning that the current RMP solution is optimal also for the MP; the corresponding value is retained as a valid lower bound for the fm-BPPSPF. If the integrality constraints of the integer problem are satisfied, then the solution is also optimal for the fm-BPPSPF, otherwisewe enter a search tree by performing branching operations as described in Subsection 4.2.3.

4.2.1 *Initialization*

In order to reduce heading-in effects, we populate the RMP with a set of columns that ensure the starting RMP to be feasible.

Our heuristic initialization approach is based on the iterative generation of columns obtained by solving subset-sum problems: the algorithm (see Pseudocode 4.2.1) generates at each iteration a set of chains having the same length, and packs the items by minimizing the residual capacity of each chain. The starting length of the chains is set to one, while the maximum allowed length is set to $|B|$.

The packing relies on a *Subset-Sum (SS)* Procedure that takes in input a set of items J, their weights w and a capacity Q, and returns the set of items $\bar{J} \subseteq J$ of largest overall weight not exceeding Q.

```
function INITRMP(I, w, B, C)
    for k = 1 ... |B| do
        J ← I
        do
            J̄ ←SS(J, w, k · C)
            add J̄ to RMP as a new column
            J ← J \ J̄
        while J ≠ ∅
    end for
end function
```

Pseudocode 4.2.1: RMP initialization algorithm

In our algorithm, the SS Procedure exploits a simple dynamic programming recursion, as described in [55]. It is easy to observe that this approach always produces a set of columns forming a feasible RMP solution. In particular, during iteration $k = |B|$, the initialization algorithm packs all the items a single chain of $|B|$ bins, thereby creating a BPPSPF solution with a number of fragmentations equal to $|B| - 1$.

4.2.2 *Pricing problem*

For each $p \in \Gamma$, the reduced cost of variable y^p is computed as

$$\pi^p = \bar{(}l^p - 1) - \sum_{i \in I} \lambda_i \cdot \bar{z}_i^p + \mu \cdot \bar{l}^p.$$

The pricing problem, that is the problem of finding the most negative reduced cost column, can be stated as follows:

$$\begin{aligned} \pi^* = \min_{p \in \Gamma} \quad & \bar{l}^p - \sum_{i \in I} \lambda_i \cdot \bar{z}_i^p + \mu \cdot \bar{l}^p - 1 && (4.2.1)\\ \text{s.t.} \quad & \sum_{i \in I} w_i \cdot \bar{z}_i^p \leqslant C \cdot \bar{l}^p \\ & 0 \leqslant \bar{l}^p \leqslant |B| \\ & \bar{z}_i^p \in \mathbb{B} && \forall i \in I \\ & \bar{l}^p \in \mathbb{N} \end{aligned}$$

Let us state the objective function of the pricing problem (4.2.1) in maximization form, and collect the coefficients of terms z_i^p and l^p

$$\pi^* = -\left(\max_{p \in \Gamma} \sum_{i \in I} \lambda_i \cdot \bar{z}_i^p - (\mu + 1) \cdot \bar{l}^p + 1 \right).$$

That is, λ_i and $(\mu + 1)$ represent the prize for packing item i and the cost for using each bin in the current chain, respectively. Therefore, the pricing problem is a variant of the 0-1 Knapsack Problem (KP) [55]. It aims to find an optimal tradeoff between the cost of using bins and the profit of including items, respecting a single aggregated capacity constraint and an upper bound on the available capacity. We indicate such a variant as the *Variable Size KP* (VSKP).

To solve the VSKP we develop the following ad-hoc pseudo-polynomial time algorithm based on the well-known dynamic programming approach for the 0-1 KP [55]: let $M(\bar{I}, c)$ be the cost of an optimal VSKP solution in which only items $\bar{I} \subseteq I$ are allowed to be selected, and exactly c units of capacity are consumed. The values $M(\bar{I}, c)$ can be recursively computed as follows:

$$\begin{aligned} M(\bar{I} \cup \{i\}, c) = & -(\mu + 1) \cdot \lceil c/C \rceil \\ & + \begin{cases} M(\bar{I}, c), \text{ if } w_i > c \\ \max\left\{ M(\bar{I}, c); M(\bar{I}, c - w_i) + \lambda_i \right\}, \text{otherwise} \end{cases} \end{aligned}$$

where $M(\emptyset, c) = +1$ for each $0 \leqslant c \leqslant |B| \cdot C$. Thus, the final cost of an optimal VSKP is

$$\pi^* = \max_{0 \leqslant c \leqslant |B| \cdot C} \{M(I, c)\}$$

It is easy to keep track of the subset of items forming an optimal solution by storing which arguments yield maxima in the above expression. The complexity of the overall procedure is $O(|B| \cdot C \cdot |I|)$.

As multiple pricing strategy, we add $|K|$ columns at each column generation iteration, corresponding to the values $M(I, k \cdot C)$ for $k = 1 \dots |K|$.

4.2.3 *Branch and bound*

When the optimal MP solution is fractional, and upper and lower bounds do not match, we check which integrality constraints are not satisfied w.r.t. the integer formulation, and we explore a search tree by means of branching.

In our case, branching is particularly involved, as the MP is prone to symmetries. We devised the following binary branching rule, in which chains are progressively defined and an integer solution is enforced through the assignment of items to the chains.

Let us suppose that a particular *head item* is defined for each chain, and that at a certain node of the branching tree, items in a set $H \subseteq I$ are selected to be *head items* of $|H|$ chains. At the root node, $H = \emptyset$; then, recursively, a branching item is either selected to be assigned to one of the chains identified by an item in H, or becomes a new head item, thereby identifying an additional chain.

PHASE 1 - ITEM ASSIGNMENT. If $H = \emptyset$, we directly skip to Phase 2. Otherwise, let $\tilde{y}^p$ be the values of each variable y^p in a fractional MP solution and, for each $i \in I$ and $h \in H$

$$t_{ih} = \sum_{p \in \Gamma} \bar{z}_i^p \cdot \bar{z}_h^p \cdot \tilde{y}^p$$

be a coefficient that indicates how much item i is packed with head item h in the fractional solution. We search for an item $\hat{i}$ and a head item $\hat{h}$ corresponding to the most fractional assignment in the current fractional solution, that is

$$(\hat{h}, \hat{i}) \in \operatorname*{argmin}_{h \in H, i \in I} \left\{ \left| t_{ih} - \frac{1}{2} \right| \right\}.$$

If $t_{\hat{i}\hat{h}}$ is fractional, then we perform binary branching: we enforce $\hat{i}$ to be always packed with $\hat{h}$ in one branch, while we forbid $\hat{i}$ to be

packed with $\hat{h}$ in the other. If instead $t_{\hat{i}\hat{h}}$ is integer, we proceed to Phase 2.

PHASE 2 - CHAIN SELECTION. If no fractional assignment of items to chains defined by H can be found, then it also holds that no column in the MP having a fixed head item is fractionally selected. In fact, if it were, such fractionally selected columns should be identical, but in our MP it is never profitable to generate the same column twice. However, it is still possible that fractional solutions arise due to splitting in additional chains for which no head item is fixed.

We check this condition as follows. We search for the most fractionally selected MP variable, that is we identify

$$\hat{p} \in \operatorname*{argmin}_{p \in \Gamma} \left\{ \left| \tilde{y}^p - \frac{1}{2} \right| \right\}.$$

If $\tilde{y}^{\hat{p}}$ is integer, then a full integer solution is found: the incumbent is possibly updated and the branching node is fathomed.

Otherwise, an arbitrary item $\hat{i}$ is selected, such that $\tilde{z}^{\hat{p}}_{\hat{i}} = 1$ and $\hat{i} \notin H$. Then we add $\hat{i}$ to the set H, we initialize $I_{\hat{i}} = \{\hat{i}\}$, and we restart branching from Phase 1.

PRICING IMPLEMENTATION. Our branching strategy alters the nature of the pricing problem. Let I_h be the set of items forced to be packed with head item h, let $W_h = \sum_{i \in I_h} w_i$ be their sum of weights, and let $\bar{I}_h$ be the set of items whose packing with h is forbidden. Let I_0 be the set of items which are not forced to be packed to any head item, such that

$$I_0 = I \setminus \bigcup_{h \in H} I_h.$$

We deal with additional constraints introduced in both branching phases by solving $|H| + 1$ VSKPs. The first VSKP aims at finding the chain without head item yielding the most negative reduced cost:

$$\pi_0 = - \max_{0 \leqslant c \leqslant |B| \cdot C} \{M(I_0, c)\}.$$

Then, we solve a VSKP for each $h \in H$, each searching for the chain with head item h yielding the most negative reduced cost:

$$\pi_h = - \sum_{i \in I_h} \lambda_i - \max_{0 \leqslant c \leqslant |B| \cdot C - W_h} \{M(I_0 \setminus \bar{I}_h, c)\}.$$

that is, we solve a VSKP where the available capacity is decreased by the weights of the items fixed in I_h, forbidding the selection of items in $\bar{I}_h$ and decreasing the final reduced cost by the sum of the prizes of the items in I_h.

We experimentally observed that, although the number of VSKP subproblems increases as the depth of the branching tree increases, the overall number of chains remains limited. Additionally, the solutions of the $|H| + 1$ VSKPs yield well diversified columns, that are in turn useful to perform more effective multiple pricing, thus speeding up the column generation process.

In particular, after preliminary experiments, we set a different multiple pricing strategy in the inner nodes of the branching tree. That is, at each iteration of column generation we add to the RMP (a) one column, corresponding to the packing defining the value of π_0 and (b) $|H|$ columns, each corresponding to the packing defining the value of $i\pi_h$ for each $h \in H$, provided they have negative reduced cost.

4.3 TACKLING THE SIZE INCREASING VARIANT

Several BPPIF variants arise in the literature and in practical applications. One of the main features changing among them is the possibility of handling overhead in item weights after each split.

First, still owing to practical applications, packing an item may introduce an overhead in its weight. This is the typical case of transmissions over packed-switching networks, in which data are split into packets that need additional headers (and trailers) to be delivered, as stated in [69].

Let ϵ be a constant representing an amount of overhead, that we suppose to satisfy $\epsilon \leqslant \min_{i \in I} w_i$. The BPPIF with size-increasing fragmentation (BPPSIF) is the variant of BPPIF in which a weight ϵ is attached to all fragments packed into bins, including those fragments corresponding to unfragmented items. We first observe that

Proposition 4.3.1. *an optimal solution to fm-BPPSIF always exists, that is primitive.*

The proof is similar to that of Theorem 3.1.1, additionally observing that a non-primitive optimal solution would just introduce more overhead than a corresponding primitive one.

Thus we adapt the chain-based BPPIF model (4.1.1) – (4.1.8), by changing constraint (4.1.4) into

$$\sum_{i \in I} (w_i + \epsilon) \cdot z_{ik} + \epsilon \cdot (l_k - 1) \leqslant C \cdot l_k \qquad \forall k \in K, \tag{4.3.1}$$

where the term $\epsilon \cdot (l_k - 1)$ is the overhead given by the chain fragmentations.

A corresponding extended formulation can also be obtained, by performing the reformulation steps described on (4.1.16) – (4.1.19), and changing only the definition of sets Γ^k. Indeed, our BPPSPF models can be obtained as a special case of BPPSIF ones by setting $\epsilon = 0$.

We remark that other overhead policies may be pertinent depending on the practical application, as in [56]. The methodology may still be valid depending on the existence of a primitive optimal solution.

Overhead handling nicely fits in our framework. In fact, Constraint (4.3.1) can be easily transposed into the pricing problem and solved as a variant of VSKP in which bins capacity is $\hat{C} = C - \epsilon$, and a capacity of ϵ is set as consumed since the beginning. Due to Constraint (4.3.1) the weight of each item i is then $\hat{w}_i = w_i + \epsilon$. Once again, when $\epsilon = 0$, the pricing algorithm is exactly the one described in Section 4.2.2.

We remark that also model (3.2.1) – (3.2.6) can be adapted to solve the size-increasing variant by changing constraint (3.2.3) into

$$\sum_{i \in I} (w_i \cdot x_{ij} + \epsilon \cdot z_{ij}) \leqslant C \qquad \forall j \in B. \tag{4.3.2}$$

However, from preliminary results we decided to implement the size-increasing variant only into the chain formulation.

4.4 EXPERIMENTAL RESULTS

We implemented our algorithms in C++, using the framework SCIP [1] version 3.0.2, keeping the default options but forcing single thread execution. In particular, that includes a full suite of general purpose primal heuristics. We also include SSH procedure of Section 3.4.2 as primal heuristic. The LP subproblems were solved using the simplex algorithm implemented in CPLEX 12.4 [36]: the framework automatically switches between primal and dual methods. We refer to our exact branch-and-price algorithm as BPCA in the remainder.

As a benchmark we considered the branch-and-cut implemented in CPLEX 12.4, using the mathematical programming models described

in Section 3.2 and Section 4.1, and keeping again default settings besides forcing single thread execution. All the tests have been performed on a PC equipped with an Intel(R) Core2 Duo CPU E6850 at 3.00 GHz and 4 GB of memory.

We tested our BPCA on the dataset for the fm-BPPSPF described in Section 3.5. We included three benchmark algorithms: CPLEX using the compact model, CPLEX using the chain-based model, and the branch-and-price algorithm described in Chapter 3. A time limit of one hour was given to each run.

In Table 4 we report for each solution the number of instances solved to proven optimality within the time limit (S), the average duality gap on the remaining ones computed as $(UB - LB)/UB$ (Gap) and the average computing time (t). Our BPCA outperforms the three benchmark algorithms by far, solving all the instances in fractions of second. It is also interesting to observe that chain-based models allow to obtain always better results when CPLEX is employed: on a few classes of instances, CPLEX using the chain-based models performs better than the BP algorithm.

4.4.1 *Dataset generation*

Indeed, the design of a dataset being challenging, statistically significant and fair at the same time turned out to be an issue on its own. First, the results obtained in Section 3.5 indicate that the ratio between item weights and capacity, and the amount of residual capacity are the features influencing most the computational behavior of both general purpose solvers and ad-hoc algorithms. Second, the behavior of general purpose solvers is strongly influenced by the number of available bins $|B|$. On the contrary, our pricing algorithms have a pseudo-polynomial time complexity in the capacity value C, and therefore such a parameter can influence the performance of BPCA. Therefore a full control on all these parameters is needed to detect regularities.

Hence, we created a new dataset as follows. We considered instances including $|I| = 20$, 50 or 100 items. The capacity C of each bin was always fixed to 1000. Then we considered three types of weight distributions: let $\bar{w}$ and $\underline{w}$ be respectively upper and lower ends of the range of possible weight values

LARGE: draw integers from a uniform distribution between $\underline{w} = 0.5 \cdot C$ and $\bar{w} = 0.9 \cdot C$

	Class		CPLEX			CPLEX chain model			BP			BPCA		
|I|	w.	cap.	S	Gap (%)	t (s)	S	Gap (%)	t (s)	S	Gap (%)	t (s)	S	Gap (%)	t (s)
10	big	tight	6	20.8	427.2	10	0.0	0.2	10	0.0	137.4	10	0.0	0.0
10	free	tight	10	0.0	36.3	10	0.0	0.0	10	0.0	0.4	10	0.0	0.0
10	small	tight	10	0.0	0.8	10	0.0	0.0	10	0.0	0.1	10	0.0	0.0
10	big	loose	10	0.0	140.4	10	0.0	0.4	10	0.0	0.2	10	0.0	0.0
10	free	loose	10	0.0	0.0	10	0.0	0.0	10	0.0	0.0	10	0.0	0.0
10	small	loose	10	0.0	0.0	10	0.0	0.0	10	0.0	0.0	10	0.0	0.0
15	big	tight	0	17.4	-	10	0.0	62.0	1	12.5	0.3	10	0.0	0.1
15	free	tight	0	56.7	-	10	0.0	8.4	6	16.7	6.0	10	0.0	0.1
15	small	tight	7	55.6	1,090.2	10	0.0	1.3	10	0.0	0.5	10	0.0	0.0
15	big	loose	0	65.0	-	8	37.5	1,231.1	10	0.0	5.0	10	0.0	0.1
15	free	loose	10	0.0	0.1	10	0.0	0.0	10	0.0	0.1	10	0.0	0.0
15	small	loose	10	0.0	0.0	10	0.0	0.0	10	0.0	0.0	10	0.0	0.0
20	big	tight	0	29.9	-	2	29.2	2,525.0	0	12.5	-	10	0.0	0.2
20	free	tight	0	82.3	-	1	57.4	2,891.4	3	16.7	124.7	10	0.0	0.1
20	small	tight	0	86.7	-	10	0.0	97.8	10	0.0	2.3	10	0.0	0.1
20	big	loose	0	93.3	-	0	63.3	-	3	16.7	1,822.9	10	0.0	0.1
20	free	loose	10	0.0	0.1	10	0.0	0.1	10	0.0	0.1	10	0.0	0.1
20	small	loose	10	0.0	0.0	10	0.0	0.0	10	0.0	0.1	10	0.0	0.0

Table 4: Solving fm-BPPSPF to proven optimality.

SMALL: draw integers from a uniform distribution between $\underline{w} = 0.1 \cdot C$ and $\bar{w} = 0.5 \cdot C$

FREE: draw integers from a uniform distribution between $\underline{w} = 0.1 \cdot C$ and $\bar{w} = 0.9 \cdot C$

We initially considered fragmentations-minimization. With respect to the residual capacity, we considered two types of instances by changing the number of available bins:

TIGHT: $|B| = \lceil \frac{\underline{w}+\bar{w}}{2} \cdot |I| \rceil$, that is the minimum expected number of bins needed to fractionally pack all the items,

LOOSE: 10% of expected residual space, that is $|B| = \lceil \frac{1.0}{0.9} \cdot \frac{\underline{w}+\bar{w}}{2} \cdot |I| \rceil$.

Finally, the weights of tight instances were generated as follows. For the first $|I| - 1$ items we set the weights to be a random integer value chosen between $\underline{w}$ and $\bar{w}$, while the last item weight was given by the difference between $C \cdot |B|$ and the sum of the previously generated weights. The generation was repeated until the last weight was between $\underline{w}$ and $\bar{w}$. The weights of loose instances were generated similarly, but setting the weight of the last item to the difference between $\frac{1.0}{0.9} \cdot C \cdot |B|$ and the sum of the previously generated weights.

We created an instance class for each combination of instance size, weight distribution and residual capacity, and generated ten instances for each class, obtaining a dataset of 180 instances. In this way, each class contains instances having homogeneous $|I|$, $|B|$, and C values.

4.4.2 *Solving the Size-Increasing variant*

In our experiments we assessed the impact of size-increasing features both on the computational behavior of our algorithms and on the final solution costs. The analysis has been performed using BPCA. No time limit was given to these test, in order to always obtain a global optimal solution. Nevertheless, no test exceeded one hour of computation. The overhead ϵ was set to be 1% of the bin capacity.

When setting large overhead ($\epsilon = 0.1 \cdot C$), the fraction of infeasible instances was too large to provide any meaningful result, since for most instances the sum of the items weight plus the overhead of each item was already greater than the available capacity. Therefore, we test instances with small overhead ($\epsilon = 0.01 \cdot C$) only. In Table 5 we report the computing time required to solve to optimality both size-preserving and size-increasing instances. For the latter we also report

the average gap between the optimal solutions values obtained with the size-preserving model (SP) and the ones obtained with the size-increasing variant (SI), computed as $(SI - SP)/SP$.

All instances having tight capacity resulted infeasible by design: it is however interesting to note that infeasibility is detected quickly by our algorithms. In the remaining instances, the penalties are in a few cases very high. The computing time showed to be not an issue: all instances were solved within five minutes, even if by introducing overhead the required CPU time slightly increased.

Class			Size Preserving	Size Increasing	
$\|I\|$	w.	cap.	t (s)	Gap (%)	t (s)
20	big	tight	0.2	-	0.1
20	free	tight	0.1	-	0.1
20	small	tight	0.1	-	0.0
20	big	loose	0.1	5.0	0.1
20	free	loose	0.1	40.0	0.1
20	small	loose	0.0	0.0	0.0
50	big	tight	8.4	-	0.9
50	free	tight	4.3	-	0.5
50	small	tight	1.5	-	0.3
50	big	loose	0.8	0.0	2.5
50	free	loose	0.7	15.0	1.2
50	small	loose	0.2	0.0	0.2
100	big	tight	235.7	-	6.1
100	free	tight	252.6	-	3.4
100	small	tight	35.6	-	1.4
100	big	loose	5.0	0.0	15.0
100	free	loose	5.5	0.0	10.5
100	small	loose	0.8	0.0	0.8

Table 5: Solving fm-BPPSIF with small overhead. Rows with a $-$ symbol in the Gap column refer to infeasible instances.

4.5 CONCLUSIONS

In this chapter we proposed a new approach to solve the fm-BPPIF: after collecting and deriving some properties of BPPIF solutions, we proposed a new mathematical programming model that avoid the use of fractional variables. Then, by means of Dantzig-Wolfe decomposition we also introduced an extended formulation. We exploited column generation techniques with ad-hoc pricing algorithms, heuristics

and implicit enumeration to design a new exact branch-and-price algorithm.

Such algorithm proved to be first of all flexible, as minor or no modifications are needed to adapt it to size preserving or size increasing overhead management policies.

Our experimental campaign revealed that state-of-art general purpose solvers like CPLEX find it beneficial to use our new models, but still fail in optimizing even instances of very small size. Instead, our algorithms proved to be very effective, being able to solve to proven optimality all the instances in our datasets in minutes of computation for any BPPIF variant, thus outperforming both CPLEX and previous approaches.

5

BIN MINIMIZATION BPPIF

We now discuss the bin-minimization variant of the BPPIF with size preserving fragmentation (bm-BPPSPF). For the ease of exposition, in Section 5.1 we describe the problem and present a few theoretical properties on the structure of the optimal solution that lead to a pure combinatorial mathematical programming model of the problem. As for the fm-BPPSPF, we then present its extended formulation obtained by applying Dantzig-Wolfe decomposition. In Section 5.2 we detail our new exact algorithm to solve it. Then, in Section 5.3 we discuss on how to extend it in order to tackle the size increasing variant. In Section 5.4 we present our experimental analysis. Finally, in Section 5.5 we summarize our results and collect some brief conclusions.

5.1 PROBLEM DEFINITION

We are given a set of items I and a set of bins B. Let w_i be the weight of each item $i \in I$, and let C be the capacity of each bin. Each item has to be fully packed, but may be split into fragments and fractionally assigned to different bins. The sum of the weights of the (fragments of) items packed into a single bin must not exceed the capacity C.

The bm-BPPSPF can be stated as the problem of packing all the items into the minimum number of bins by performing at most $\mathcal{F}$ fragmentations. It can be formalized as follows:

$$\min \quad \sum_{j \in B} u_j \tag{5.1.1}$$

$$\text{s.t.} \quad \sum_{j \in B} x_{ij} = 1 \qquad \forall i \in I \tag{5.1.2}$$

$$\sum_{i \in I} w_i \cdot x_{ij} \leqslant C \cdot u_j \qquad \forall j \in B \tag{5.1.3}$$

$$\sum_{\substack{i \in I \\ j \in B}} z_{ij} - |I| \leqslant \mathcal{F} \tag{5.1.4}$$

$$x_{ij} \leqslant z_{ij} \qquad \substack{\forall i \in I \\ \forall j \in B} \tag{5.1.5}$$

$$0 \leqslant x_{ij} \leqslant 1 \qquad \substack{\forall i \in I \\ \forall j \in B} \tag{5.1.6}$$

$$z_{ij} \in \{0, 1\} \qquad \substack{\forall i \in I \\ \forall j \in B} \tag{5.1.7}$$

$$u_j \in \{0, 1\} \qquad \forall j \in B \tag{5.1.8}$$

where each variable x_{ij} represents the fraction of item i packed into bin j, each binary variable z_{ij} is 1 if any fragment of item i is packed into bin j, and each binary variable u_j is 1 if bin j is open.

The objective function (5.1.1) minimizes the number of open bins. Constraints (5.1.2) ensure that each item is fully packed. Constraints (5.1.3) have a double effect: they forbid the assignment of items to bins that are not open and ensure that the capacity of each open bin is not exceeded. Constraint (5.1.4) ensures that the packing is performed with at most $\mathcal{F}$ fragmentations. Constraints (5.1.5) enforce consistency between variables, so that no fragment x_{ij} of each item i is packed into bin j unless z_{ij} is set to 1.

Observation 5.1.1. *Given any BPPIF solution, the number of fragmentations is equal to the overall number of fragments minus the number of items.*

According to the literature [[69]]:

Theorem 5.1.1. *Any instance of bm-BPPSPF has an optimal solution which is primitive.*

It is easy to prove that such a structure directly influences the cost of a packing by representing a primitive solution through the construction of its BPG.

Observation 5.1.2. *The cost of a primitive solution is the sum of the length of all the chains in its BPG.*

In fact, the length of a chain is the number of bins in the corresponding path of the BPG. Hence, the sum of all chain lengths is the overall

number of bins used in a certain solution, and thus the cost of the packing. Furthermore, similarly to Proposition 4.1.1, packing all item of a chain by using NF_f does not affect the cost of the solution, since the length of each chain is fixed and thus the overall number of used bins does not change.

By exploiting these properties, we can model the bm-BPPSPF as the problem of optimally packing items into chains instead of bins.

Let K be a set of chains. Let l_k be a variable representing the length of each $k \in K$, that is, each $k \in K$ includes a set of l_k bins, involves $l_k - 1$ item splits and provides an overall capacity of $l_k \cdot C$. Model (5.1.1) – (5.1.8) can be reformulated as follows:

$$\min \quad \sum_{k \in K} l_k \tag{5.1.9}$$

$$\text{s.t.} \quad \sum_{k \in K} z_{ik} = 1 \qquad \forall i \in I \tag{5.1.10}$$

$$\sum_{k \in K} l_k - v_k \leqslant \mathcal{F} \tag{5.1.11}$$

$$\sum_{i \in I} w_i \cdot z_{ik} \leqslant C \cdot l_k \qquad \forall k \in K \tag{5.1.12}$$

$$v_k \leqslant l_k \qquad \forall k \in K \tag{5.1.13}$$

$$z_{ik} \in \mathbb{B} \qquad \begin{matrix} \forall i \in I \\ \forall k \in K \end{matrix} \tag{5.1.14}$$

$$l_k \in \mathbb{N} \qquad \forall k \in K \tag{5.1.15}$$

$$v_k \in \mathbb{B} \qquad \forall k \in K \tag{5.1.16}$$

Here, each binary variable z_{ik} is set to 1 if item i is packed into chain k, and each binary variable v_k is set to 1 if chain k contains at least one item.

The objective function (5.1.9) minimizes the number of used bins. Each item can be split among bins in the same chain, but no fractional assignment of items to different chains is allowed: constraints (5.1.10) impose that each item is fully packed in a single chain. Constraints (5.1.12) ensure that the capacity of each chain is not exceeded, and constraint (5.1.11) guarantees that at most $\mathcal{F}$ fragmentations are performed. Constraints (5.1.13) enforce consistency between variables so that a chain is used only if its length is at least one.

We remark that a solution of (5.1.9) – (5.1.16) encodes no information on which items are split, nor on which items are packed into the same bin of each chain. In fact, due to Proposition 4.1.1, it is possible to obtain a feasible solution of bm-BPPSPF starting from any feasible

solution of (5.1.9)–(5.1.16) with post processing, by applying the NF_f algorithm on each chain independently.

From a continuous relaxation point of view, neither formulation (5.1.1)–(5.1.8) nor formulation (5.1.9)–(5.1.16) offer a significant lower bound: in the first model, it is possible to fix $|B| = |I|$, and pack each item i into bin $j = i$ fixing each $u_i = w_i/C$. The corresponding objective function value is $\sum_j u_j = \sum_i w_i/C$, yielding a trivial lower bound. Likewise in the chain model, by fixing $|K| = |I|$ and $l_i = w_i/C$.

5.1.1 *Extended formulation*

To obtain a significat lower bound we reformulate the problem exploiting Dantzig-Wolfe decomposition method [30]. The process of such a reformulation is very similar to the process described in Section 4.1.1, and thus we present the final Master Problem (MP), that is:

$$\min \quad \sum_{p\in\Gamma} \bar{l}^p \cdot y^p \tag{5.1.17}$$

$$\text{s.t.} \quad \sum_{p\in\Gamma} \bar{z}_i^p \cdot y^p \geqslant 1 \qquad \forall i\in I \tag{5.1.18}$$

$$-\sum_{p\in\Gamma} (\bar{l}^p - 1) \cdot y^p \geqslant -\mathcal{F} \tag{5.1.19}$$

$$y^p \geqslant 0 \qquad \forall p\in\Gamma \tag{5.1.20}$$

Observation 5.1.3. *The lower bound provided by the MP dominates that given by the continuous relaxation of model (5.1.9) – (5.1.16).*

The observation directly follows from the Dantzig-Wolfe decomposition principle. We further observe that, although the continuous relaxations of models (5.1.1) – (5.1.8) and (5.1.9) – (5.1.16) are equivalent, their Dantzig-Wolfe decompositions are not. In particular, as for fm-BPPSPF:

Proposition 5.1.1. *The lower bound provided by the MP dominates that obtained through Dantzig-Wolfe decomposition of model (5.1.1)–(5.1.8);*

5.2 ALGORITHMS

As for the algorithm described in Section 4.2, we solve the extended formulation by means of column generation techniques. We initialize the RMP with a small subset of columns including valid dual cuts (see Subsection 5.2.1) and solve the RMP to optimality. We show in

Subsection 5.2.2 that the pricing problem is still a variant of the 0-1 Knapsack Problem (KP) that we solve with the same algorithm described in Subsection 4.2.2. After solving the MP to optimality, if the solution violates integrality constraints of the integer problem, we enter a search tree by performing the same branching operations used for the fm-BPPSPF and described in Subsection 4.2.3.

5.2.1 *Initialization*

In order to reduce heading-in effects [80], we populate the RMP with two sets of columns. The first one consists of chains of different length that pack all the items as in Subsection 4.2.1. The second one is composed by polynomial families of dual cuts; as a side effect, they help to stabilize and to speedup the overall column generation process.

SUBSET-SUM COLUMNS. Our initialization approach is based on the iterative generation of columns obtained by solving several Subset-Sum problems: the algorithm (see Pseudocode 5.2.1) generates at each iteration a set of chains having the same length, and packs the items by minimizing the residual capacity of each chain. The starting length of the chains is set to one, while the maximum allowed length is set to $\mathcal{F}$.

The packing relies on a *Subset-Sum (SS)* Procedure that takes in input a set of items J, their weights w and a capacity Q, and returns the set of items $\bar{J} \subseteq J$ of largest overall weight not exceeding Q.

```
function INITRMP(I, w, F, C)
    for k = 1 ... F + 1 do
        J ← I
        do
            J̄ ←SS(J, w, k · C)
            add J̄ to RMP as a new column
            J ← J \ J̄
        while J ≠ ∅
    end for
end function
```

Pseudocode 5.2.1: RMP initialization algorithm

In our algorithm, the SS Procedure exploits a simple dynamic programming recursion, as described in [55]. This approach always produces a set of columns forming a feasible RMP solution. In particular, during iteration $k = 1$, the initialization algorithm packs all the items

in chains of single bins, thereby creating a BPPSPF solution with no fragmentations.

DUAL CUTS FOR BPPSPF. Then we restrict the dual space of the MP, in order to obtain optimal MP solutions faster. We exploit a variant of the family of dual cuts proposed in [76] for the Cutting-Stock Problem. Let λ and μ be the vectors of non negative dual variables corresponding to constraints (5.1.18) and (5.1.19), respectively. We formulate the following proposition:

Proposition 5.2.1. *For each pair of subsets* S *and* T *of* I *such that*

$$\sum_{i \in S} w_i \leqslant \sum_{i \in T} w_i$$

no optimal dual solution violates the following inequalities:

$$\sum_{i \in S} \lambda_i \leqslant \sum_{i \in T} \lambda_i \tag{5.2.1}$$

In fact, if any such inequality were violated, all basic columns of the MP representing chains including T can be replaced by columns where items in T are removed and replaced by items in S, as these still encode feasible chains. The reduced cost of these new columns would be negative, thus contradicting the optimality of the solution.

Intuitively, these inequalities encode the following condition: an optimal solution always exists, in which subsets of small weight yield less dual contribution to the reduced costs. Moreover, the linear combination of dual cuts (5.2.1) and columns in the RMP, gives origin to new patterns that may avoid the generation of further columns.

From an implementation point of view, each dual cut is represented by a column p with $y^p \geqslant 0$, in which $z_i^p = 1$ for each $i \in S$ and $z_i^p = -1$ for each $i \in T$. However, the number of these valid dual cuts grows exponentially with the number of items in I. Very recent contributions show that in particular cases their dynamic generation is appealing [45]. Instead, we found it useful to focus on sets S and T of small cardinality, considering two cases in which $|S| = |T| = 1$, and $|S| = 2$ and $|T| = 1$, and we add the corresponding columns to the RMP before the execution of the column generation process. Preliminary results showed that such approach reduces by 50% the average number of column generation iterations needed to compute a valid lower bound. Also, computing times are reduced by 20%.

5.2.2 *Pricing problem*

For each $p \in \Gamma$, the reduced cost of variable y^p is computed as

$$\pi^p = \bar{l}^p - \sum_{i \in I} \lambda_i \cdot \bar{z}_i^p + \mu \cdot (\bar{l}^p - 1).$$

The pricing problem, that is the problem of finding the most negative reduced cost column, can be stated as follows:

$$\begin{aligned} \pi^* = \min_{p \in \Gamma} \quad & \bar{l}^p - \sum_{i \in I} \lambda_i \cdot \bar{z}_i^p + \mu \cdot (\bar{l}^p - 1) && (5.2.2) \\ \text{s.t.} \quad & \sum_{i \in I} w_i \cdot \bar{z}_i^p \leqslant C \cdot \bar{l}^p \\ & 0 \leqslant \bar{l}^p - 1 \leqslant \mathcal{F} \\ & \bar{z}_i^p \in \mathbb{B} && \forall i \in I \\ & \bar{l}^p \in \mathbb{N} \end{aligned}$$

Let us state the objective function of the pricing problem (4.2.1) in maximization form, and collect the coefficients of terms z_i^p and l^p

$$\pi^* = -\max_{p \in \Gamma} \sum_{i \in I} \lambda_i \cdot \bar{z}_i^p - (\mu + 1) \cdot \bar{l}^p + \mu.$$

That is, λ_i and $(\mu + 1)$ represent the prize for packing item i and the cost for using each bin in the current chain, respectively. Therefore, the pricing problem is still a Variable Size KP (VSKP) introduced in Subsection 4.2.2. The only difference to the previous algorithm is that in this case we have a maximum capacity of $\mathcal{F} \cdot C$, and that the cost of the initial recursive function is $M(\emptyset, c) = -\mu$ for each $0 \leqslant c \leqslant \mathcal{F} \cdot C$. Thus, the final cost of an optimal VSKP is

$$\pi^* = \max_{0 \leqslant c \leqslant \mathcal{F} \cdot C} \{M(I, c)\}.$$

It is easy to keep track of the subset of items forming an optimal solution by storing which arguments yield maxima in the above expression. The complexity of the overall procedure is $O(\mathcal{F} \cdot C \cdot |I|)$.

After preliminary experiments, we found beneficial to add $|K|$ columns at each column generation iteration, corresponding to the values $M(I, k \cdot C)$ for $k = 1 \ldots |K|$.

5.2.3 *Primal Heuristics*

When the column generation process is over, the optimal MP solution can be fractional. In that case we run primal (upper bounding) heuristics: if the upper and lower bounds match, global optimality for the bm-BPPSPF is proved. In order to find good integer solutions quickly, we developed the following Iterative Subset-Sum Heuristic (ISSH), that is built on the SSH heuristic for fm-BPPSPF proposed in Subsection 3.4.2. The idea is to iteratively fix the number of open bins and pack all the items by solving a fm-BPPSPF, until a feasible solution is reached, that packs all items with a number of fragmentations not exceeding $\mathcal{F}$. The algorithm is detailed in Pseudocode 5.2.2.

function ISSH($I, w, \mathcal{F}, C$)
  $B \leftarrow \lceil \sum_{i \in I} w_i / C \rceil$
  loop
    $P, \tilde{\mathcal{F}} \leftarrow$ SSH(I, B, w, C)
    if $\tilde{\mathcal{F}} \leqslant \mathcal{F}$ **then**
      return P, B
    else
      $B \leftarrow B + 1$
    end if
  end loop
end function

Pseudocode 5.2.2: ISSH heuristic

The ISSH makes use of the SSH procedure that takes as arguments the set of items I, the number of bins B, the vector of weights w and the capacity C, and returns a set of chains P and the number of fragmentations $\tilde{\mathcal{F}}$. The algorithm always ends with a feasible solution for the bm-BPPSPF, since it is always possible to find a feasible packing that uses $|I|$ bins.

Additionally, we apply several general purpose MILP heuristics to be run on MP fractional solutions, that are included in the framework we use in our implementation; more details are reported in Section 5.4.

5.3 TACKLING SIZE INCREASING VARIANT

As for the fm-BPPIF, the bm-BPPIF with size-increasing fragmentation (bm-BPPSIF) is the variant of bm-BPPIF in which a weight ϵ is

attached to all fragments packed into bins, including those fragments corresponding to unfragmented items. According to literature [69]:

Proposition 5.3.1. *Each solution of bin-minimization BPPSIF has a primitive solution.*

Thus we adapt the chain-based BPPIF model (5.1.9) – (5.1.16), by changing constraint (5.1.12) into constraint 4.3.1.

A corresponding extended formulation can also be obtained as for the model (5.1.17) – (5.1.20), and changing only the definition of sets Γ^k. Indeed, our BPPSPF models can be obtained as a special case of BPPSIF ones by setting $\epsilon = 0$.

5.4 EXPERIMENTAL RESULTS

As in Section 4.4, we implemented our algorithms in C++, using the framework SCIP [1] version 3.0.2 with the same default parameters. Instead of the SSH procedure, we used the ad-hoc ISSH procedure described in Subsection 5.2.3. Still, the LP subproblems were solved using the simplex algorithm of CPLEX 12.4 [36]. We refer to our exact branch-and-price algorithm as BPA in the remainder.

As a benchmark we considered the branch-and-cut implemented in CPLEX 12.4, using the mathematical programming models described in Section 5.1, and keeping again default settings besides forcing single thread execution. All tests have been performed on a PC equipped with an Intel(R) Core2 Duo CPU E6850 at 3.00 GHz and 4 GB of memory.

We used the dataset described in Section 4.4 in which we fixed the maximum number of allowed fragmentations to $\mathcal{F} = \lfloor \mathcal{F}^*/2 \rfloor$, where $\mathcal{F}^*$ is the optimal solution of the corresponding fragmentations-minimization instance. In fact, we found out instances with higher values of $\mathcal{F}$ to be trivial for our algorithms.

5.4.1 *Root lower bound*

In a first round of experiments we compared the efforts for obtaining a lower bound for bm-BPPSPF problems, stopping at the root node of the branching tree with either CPLEX using the compact model, CPLEX using the chain-based model, and our branch-and-price algorithm. In Table 6 we report, for each instance class and for each method, the time spent at the root node, and the average gap $(B^* -$

$LB)/B^*$ between the corresponding lower bound LB and the optimal bm-BPPSPF solution value B^*. The results show that the chain-based model is in general the fastest way to get a lower bound, but the BPA always provides the best gap, which usually is also the value of B^*.

Class			Compact model		Chain model		BPCA	
$\|I\|$	w.	cap.	Gap (%)	t (s)	Gap (%)	t (s)	Gap (%)	t (s)
20	big	tight	7.8	0.5	7.8	0.0	0.0	0.2
20	free	tight	9.1	0.3	9.1	0.0	0.0	0.1
20	small	tight	14.3	0.2	14.3	0.0	0.0	0.1
20	big	loose	16.7	0.6	16.7	0.1	0.0	0.1
20	free	loose	9.8	0.5	9.8	0.0	0.0	0.1
20	small	loose	0.0	0.2	0.0	0.0	0.0	0.1
50	big	tight	12.5	4.5	12.5	0.3	0.0	0.7
50	free	tight	3.8	4.0	3.8	0.3	0.4	1.5
50	small	tight	3.1	3.2	3.1	0.2	0.0	0.9
50	big	loose	20.0	5.6	20.0	0.4	0.0	0.8
50	free	loose	5.7	5.2	5.7	0.3	0.0	0.8
50	small	loose	0.0	3.1	0.0	0.2	0.0	0.9
100	big	tight	12.5	8.5	12.5	2.9	0.0	3.6
100	free	tight	2.0	8.0	2.0	2.2	0.0	7.8
100	small	tight	0.0	6.7	0.0	1.4	0.0	4.3
100	big	loose	21.3	9.2	21.3	3.0	0.0	4.6
100	free	loose	5.0	8.5	5.0	2.0	0.0	6.7
100	small	loose	0.0	6.3	0.0	1.5	0.0	14.1

Table 6: Computing lower bounds at the root node

5.4.2 *Root upper bound*

In a second round of experiments we compared the quality of the upper bounds for bm-BPPSPF obtained at the root node of the branching tree by CPLEX heuristics using either compact or chain-based models, and by our branch-and-price algorithm. In Table 7 we report for each instance class and for each method, the average gap $(UB - B^*)/UB$ between the corresponding best upper bound UB and the optimal bm-BPPSPF solution value B^*. The results show that at root node, our algorithm always provides a better upper bound, which is usually already an optimal solution. For what concerns the use of CPLEX, chain-based models clearly give better upper bounds than the compact model. BPA always offers the most accurate results.

	Class		Compact model		Chain model		BPCA	
|I|	w.	cap.	Gap (%)	t (s)	Gap (%)	t (s)	Gap (%)	t (s)
20	big	tight	19.4	0.5	6.1	0.0	0.6	0.2
20	free	tight	2.4	0.3	7.0	0.0	0.0	0.1
20	small	tight	7.2	0.2	0.0	0.0	0.0	0.1
20	big	loose	50.0	0.6	3.2	0.1	0.0	0.1
20	free	loose	29.5	0.5	9.7	0.0	0.0	0.1
20	small	loose	6.3	0.2	1.3	0.0	0.0	0.1
50	big	tight	81.4	4.5	5.8	0.3	0.0	0.7
50	free	tight	20.9	4.0	11.5	0.3	0.4	1.5
50	small	tight	21.7	3.2	9.2	0.2	3.1	0.9
50	big	loose	100.0	5.6	0.2	0.4	0.0	0.8
50	free	loose	100.0	5.2	12.0	0.3	1.4	0.8
50	small	loose	11.5	3.1	8.0	0.2	0.0	0.9
100	big	tight	100.0	8.5	23.3	2.9	0.0	3.6
100	free	tight	58.1	8.0	48.2	2.2	1.1	7.8
100	small	tight	57.8	6.7	19.4	1.4	3.2	4.3
100	big	loose	100.0	9.2	10.0	3.0	0.0	4.6
100	free	loose	100.0	8.5	17.9	2.0	1.8	6.7
100	small	loose	30.9	6.3	36.8	1.5	1.3	14.1

Table 7: Computing upper bounds at the root node

5.4.3 *Solving bm-BPPSPF to proven optimality*

Third, we ran a comparison between CPLEX on both compact models and chain-based ones, and our BPA in solving bm-BPPSPF instances to proven optimality. A time limit of one hour was given to each run.

In Table 8 we report for each method the number of instances solved to proven optimality within the time limit (S), the average duality gap on the remaining ones computed as $(UB - LB)/UB$ (Gap), and the average computing time (t).

Our BPA solves all the 180 instances, while CPLEX solves only 21 and 29 of them, using the compact and the chain-based models, respectively. This test confirms previous results on the hardness of BPPIF for generic solvers. Chain-based models perform better than compact ones, always yielding a smaller optimality gap.

5.4.4 *Solving Size-Increasing variants*

In the last round of experiments we assessed the impact of size-increasing features both on the computational behavior of our algorithm and on the final solution costs. The analysis has been performed using BPA. No time limit was given to these test, in order to always obtain a global optimal solution. Nevertheless, no test exceeded one hour of computation. The overhead ϵ was set to be 1% of the bin capacity.

In Table 9 we report the average gap between the optimal solutions values obtained on the bm-BPPIF with the size preserving model (SP) and the ones obtained with the size-increasing variant (SI), computed as (SI - SP)/SP.

From the results we observe that overhead mildly worsens the quality of the solutions, while the execution times are actually insensitive of the overhead management policy.

As a computational stress test, we repeated the experiment by increasing the overhead ϵ to 10% of the bin capacity, although in practical applications such a large overhead would not be meaningful. In Table 10 we report our results.

Still, no optimization required more than one hour of computing time. More aggressive overhead settings do not necessarily result in more difficult problems from the computing time point of view. The solutions quality, instead, is highly penalized by the large overhead: solutions are in a few cases more than 30% worse than their size-preserving counterpart. We also observed that the impact of high

	Class		Compact model			Chain model			BPCA		
|I|	w.	cap.	S	Gap (%)	t (s)	S	Gap (%)	t (s)	S	Gap (%)	t (s)
20	big	tight	0	10.2	-	1	8.0	3,448.9	10	0.0	0.2
20	free	tight	0	9.1	-	0	9.1	-	10	0.0	0.1
20	small	tight	1	14.3	335.9	0	14.3	-	10	0.0	0.1
20	big	loose	0	16.7	-	0	16.7	-	10	0.0	0.1
20	free	loose	0	9.8	-	4	9.0	0.1	10	0.0	0.1
20	small	loose	10	0.0	0.2	10	0.0	0.1	10	0.0	0.1
50	big	tight	0	24.1	-	0	12.5	-	10	0.0	0.7
50	free	tight	0	6.0	-	0	3.8	-	10	0.0	1.6
50	small	tight	0	6.3	-	0	6.3	-	10	0.0	0.9
50	big	loose	0	36.0	-	0	20.0	-	10	0.0	0.8
50	free	loose	0	7.4	-	0	5.7	-	10	0.0	0.8
50	small	loose	10	0.0	22.2	10	0.0	0.3	10	0.0	0.9
100	big	tight	0	75.6	-	0	12.5	-	10	0.0	3.5
100	free	tight	0	8.4	-	0	2.3	-	10	0.0	11.2
100	small	tight	0	5.9	-	0	3.2	-	10	0.0	9.4
100	big	loose	0	100.0	-	0	21.3	-	10	0.0	4.6
100	free	loose	0	9.2	-	0	5.0	-	10	0.0	7.6
100	small	loose	0	3.8	-	4	3.2	1,321.6	10	0.0	53.2

Table 8: Solving bm-BPPSPF to proven optimality.

	Class		Size Preserving	Size Increasing	
|I|	w.	cap.	t (s)	Gap (%)	t (s)
20	big	tight	0.2	2.0	0.1
20	free	tight	0.1	0.0	0.1
20	small	tight	0.1	0.0	0.1
20	big	loose	0.1	0.0	0.1
20	free	loose	0.1	2.7	0.1
20	small	loose	0.1	0.0	0.1
50	big	tight	0.7	0.0	1.0
50	free	tight	1.6	0.0	1.9
50	small	tight	0.9	3.3	1.1
50	big	loose	0.8	0.0	0.9
50	free	loose	0.8	3.3	0.8
50	small	loose	0.9	0.0	2.5
100	big	tight	3.5	0.0	5.8
100	free	tight	11.2	2.0	11.2
100	small	tight	9.4	3.3	9.0
100	big	loose	4.6	0.0	5.7
100	free	loose	7.6	1.9	7.4
100	small	loose	53.2	3.3	52.8

Table 9: Solving bm-BPPSIF with small overhead.

Class			Size Preserving	Size Increasing	
|I|	w.	cap.	t (s)	Gap (%)	t (s)
20	big	tight	0.2	18.5	0.1
20	free	tight	0.1	18.2	0.1
20	small	tight	0.1	28.6	0.1
20	big	loose	0.1	5.6	0.1
20	free	loose	0.1	26.1	0.1
20	small	loose	0.1	30.0	0.1
50	big	tight	0.7	12.5	1.2
50	free	tight	1.6	23.5	1.1
50	small	tight	0.9	35.6	0.8
50	big	loose	0.8	4.0	0.8
50	free	loose	0.8	22.6	0.6
50	small	loose	0.9	31.3	1.0
100	big	tight	3.5	11.1	16.2
100	free	tight	11.2	23.7	8.3
100	small	tight	9.4	36.7	12.6
100	big	loose	4.6	3.3	5.2
100	free	loose	7.6	22.5	4.0
100	small	loose	53.2	34.3	20.0

Table 10: Solving bm-BPPSIF with large overhead.

overhead increases as the average item size decrease. In fact, for small items, an overhead of $\epsilon = 0.1 \cdot C$ means an average growth of one third of each item.

5.5 CONCLUSIONS

In this chapter we tackled the bin minimization variant of the BPPIF.

After extending properties from the fm-BPPIF, we proposed new mathematical programming models that avoid the use of fractional variables. We introduced an extended formulation obtained through a Dantzig-Wolfe decomposition and exploited column generation techniques with ad-hoc pricing algorithms, dual cuts, heuristics and implicit enumeration to design an exact branch-and-price algorithm.

As for the fm-BPPIF, we proved that also for the bm-BPPIF we can extend our approach to handle size increasing variants.

Our experimental study revealed that for the bm-BPPIF, state-of-art general purpose solver CPLEX fails in optimizing instances of very small size even with our new models. Instead, our branch-and-price algorithm completely outperformed general purpose solvers, solving to optimality all the instances in our dataset within seconds.

6

FRAGMENTED ITEM MINIMIZATION BPPIF

We now introduce a new variant of the BPPIF, where a cost is paid whenever an item is split, no matter how many times. Let us consider a delivery problem in which customers demands may be split among multiple vehicles. Indeed, customers may perceive multiple visits as a lower quality of such delivery service and an operator may want to minimize the number of unsatisfied customers. In such scenario the fragmented item-minimization BPPIF (fim-BPPIF) may arise as an operative problem to minimize the number of customers whose demands are split.

In Section 6.1 we propose a formalization of the fim-BPPSPF by a mathematical programming model, while in Section 6.2 we reduce such a variant to a pure combinatorial optimization problem. In Section 6.3 we report experimental results obtained with the two different approaches. Finally, we conclude in Section 6.4.

6.1 MATHEMATICAL FORMULATION

We are given a set of items I and a set of bins B. Let w_i be the weight of each item $i \in I$ and let C be the capacity of each bin. Each item has to be fully packed, but it may be split into fragments and fractionally assigned to different bins. The sum of the weights of the (fragments of) items packed into a single bin must not exceed C.

The fim-BPPSPF can be stated as the problem of packing all the items minimizing the number of fragmented items, and can be formalized by the following mathematical programming model:

$$\min \quad \sum_{i \in I} f_i \tag{6.1.1}$$

$$\text{s.t.} \quad \sum_{j \in B} x_{ij} = 1 \qquad \forall i \in I \tag{6.1.2}$$

$$\sum_{i \in I} w_i \cdot x_{ij} \leqslant C \qquad \forall j \in B \tag{6.1.3}$$

$$x_{ij} \leqslant z_{ij} \qquad \begin{matrix} \forall i \in I \\ \forall j \in B \end{matrix} \tag{6.1.4}$$

$$z_{ij} + z_{ik} \leqslant 1 + f_i \qquad \begin{matrix} \forall i \in I \\ \forall j,k \in B \end{matrix} \tag{6.1.5}$$

$$0 \leqslant x_{ij} \leqslant 1 \qquad \begin{matrix} \forall i \in I \\ \forall j \in B \end{matrix} \tag{6.1.6}$$

$$z_{ij} \in \{0, 1\} \qquad \begin{matrix} \forall i \in I \\ \forall j \in B \end{matrix} \tag{6.1.7}$$

$$f_i \in \{0, 1\} \qquad \forall i \in I \tag{6.1.8}$$

where each variable x_{ij} represents the fraction of item i packed into bin j, each binary variable z_{ij} is 1 if any fragment of item i is packed into bin j, and each variable f_i is 1 if item i is assigned to more than one bin, and thus it is fragmented.

The objective function (6.1.1) minimizes the number of fragmented items. Constraints (6.1.2) and (6.1.3) ensure respectively that each item is fully packed and that the capacity of each bin is not exceeded. Constraints (6.1.4) enforce consistency between variables, so that no fragment x_{ij} of each item i is packed into bin j unless z_{ij} is set to 1. Constraints (6.1.5) impose that if an item is packed in more than one bin, the corresponding variable f_i is set to 1.

Contrary to the previous variants, we know:

Observation 6.1.1. *In the fim-BPPSPF it does not always exist an optimal solution that is primitive.*

In fact let me consider the following example: we are given an instance with 6 items and 4 bins. The vector w of the items weights is $w = (2, 2, 4, 4, 4, 4)$, while the capacity of each bin is 5. Indeed, it is never possible to pack all items without split, and a primitive optimal solution is shown in Figure 7. In such solution we have two fragmentations and two fragmented items, with items 1 and 2 that are split. Instead, we could achieve a better solution as depicted in Figure 8. Such solution has only item 6 fragmented, an it is also an optimal solution.

Corollary 6.1.1. *An optimal solution to fm-BPPSPF is not necessarily an optimal solution to the fim-BPPSPF, and vice versa.*

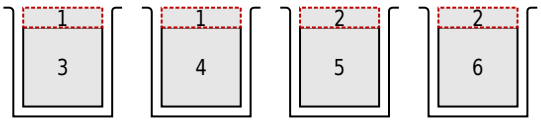

Figure 7: Example of primitive solution with 2 fragmentations and 2 fragmented items.

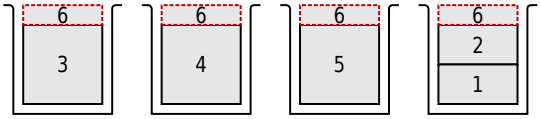

Figure 8: Example of non primitive solution with 3 fragmentations and only 1 fragmented item.

6.2 PROBLEM REDUCTION

We now show how to reduce the fim-BPPSPF to a pure combinatorial problem in order to avoid the fractional component and improve the performance while solving it with a MIP solver.

Let suppose that an optimal solution to the fim-BPPSPF is given with a mapping function $\chi : I \times B \to \{0, 1\}$ indicating if an item $i \in I$ if fully packed into bin $j \in B$.

Observation 6.2.1. *Given the function χ of an optimal packing, it is possible to obtain the complete optimal solution in polynomial time.*

In fact, considering a solution in which only non-fragmented items are given, for each bin j we can compute the residual capacity as $r_j = C - \sum_{i \in I} \mu(i, j) \cdot w_i$. Then, the fragmented items can be packed by using the NF_f algorithm on a set of bins, each one with residual capacity r_j. The resulting packing is feasible, since no overhead is introduced, and it is optimal, since no additional item is split.

Theorem 6.2.1. *Given an instance of the fim-BPPSPF, it can be reduced to a Multiple Knapsack Problem (MKP) [55] instance.*

Proof. Let $\bar{f}_i = 1 - f_i$ be the variable that is 1 if item i is not fragmented, and 0 otherwise. The objective function (6.1.1) can be rewritten as

$$\min \quad \sum_{i \in I} f_i = \sum_{i \in I} 1 - \bar{f}_i = |I| - \sum_{i \in I} \bar{f}_i.$$

We rewrite it in maximization form as

$$|I| - \max \sum_{i \in I} \bar{f}_i$$

that is the objective function that maximize the number of non-fragmented items.

Also, because Observation 6.2.1, we can solve the problem considering non-fragmented items only: x_{ij} variables can be substituted in the model by variables z_{ij} because each item is always fully packed into a single bin and Constraints (6.1.2) can be rewritten in $\leqslant$ from, since we allow some items to be excluded from the packing. The model (6.1.1)–(6.1.8) can be rewritten as

$$\max \quad \sum_{\substack{i\in I\\ j\in B}} z_{ij} \tag{6.2.1}$$

$$\text{s.t.} \quad \sum_{j\in B} z_{ij} \leqslant 1 \qquad \forall i\in I \tag{6.2.2}$$

$$\sum_{i\in I} w_i \cdot z_{ij} \leqslant C \qquad \forall j\in B \tag{6.2.3}$$

$$z_{ij} \in \{0,1\} \qquad \substack{\forall i\in I\\ \forall j\in B} \tag{6.2.4}$$

that is a MKP where each item i has a prize $p_i = 1$ when packed.

Solving such a problem on an instance of the fim-BPPSPF leads to a partial solution with non-fragmented items only. A complete solution can be obtained by applying NF_f algorithm as mentioned in Observation 6.2.1. □

Le us remark that such approach does not stand for the size increasing variant. In fact, when packing the fragmented items it may happen that the available capacity is exceeded, due to the additional overhead. However, if a feasible solution is found, then it is optimal.

6.3 EXPERIMENTAL ANALYSIS

We tested the two approaches using CPLEX 12.4 to solve MIPs: we kept default options but the number of threads was fixed to 1. All the tests have been performed on a PC equipped with an Intel(R) Core i7-2640M at 2.80 GHz and 8 GB of memory.

The dataset used for the tests is the same as the one used in Section 4.4.

Table 11 reports the results obtained allowing one hour of maximum computation: for both formulations we report the number of instances solved to optimality, the average gap $(UB - LB)/UB$ between the corresponding lower bound LB and the upper bound UB when the instance reach the time limit, and the average computing time for in-

stances solved to optimality. In our tests the MKP reformulation outperforms the fm-BPPSPF formulation by solving more than double of instances.

Class			fim-BPPSPF			MKP		
|I|	w.	cap.	S	Gap (%)	t (s)	S	Gap (%)	t (s)
20	big	tight	0	96.7	-	10	0.0	0.0
20	free	tight	8	100.0	114.0	10	0.0	0.0
20	small	tight	9	100.0	290.1	10	0.0	123.6
20	big	loose	0	100.0	-	10	0.0	0.0
20	free	loose	10	0.0	195.6	10	0.0	0.0
20	small	loose	10	0.0	0.0	10	0.0	0.0
50	big	tight	0	100.0	-	10	0.0	0.0
50	free	tight	0	100.0	-	5	70.0	1.3
50	small	tight	0	100.0	-	0	100.0	-
50	big	loose	0	100.0	-	10	0.0	0.0
50	free	loose	7	100.0	341.4	8	100.0	0.2
50	small	loose	10	0.0	2.1	10	0.0	0.0
100	big	tight	0	100.0	-	10	0.0	0.1
100	free	tight	0	100.0	-	2	84.4	15.1
100	small	tight	0	100.0	-	0	100.0	-
100	big	loose	0	100.0	-	10	0.0	0.1
100	free	loose	0	100.0	-	10	0.0	1.6
100	small	loose	10	0.0	54.8	10	0.0	0.2

Table 11: Results obtained within a time limit of one hour per instance

6.4 CONCLUSION

In this chapter we tackle a variant of the Bin Packing Problem with Item Fragmentation in which a cost is paid for each item split. We performed a theoretical investigation to devise a new approach to solve the fim-BPPSPF, reducing such problem to a Multiple Knapsack Problem, a pure combinatorial problem well-studied in literature.

Our experimental campaign revealed that with our approach, the state-of-the-art general purpose solver CPLEX is already able to solve instances with up to 100 items, while considering the fractional component lead to poor results. This is a further proof that problems with fractional resources are still hard to handle and that just by removing the fractional component, the problem comes easier to solve.

Part II

ROUTING PROBLEMS WITH PACKING ISSUES

7

SPLIT PICKUP AND SPLIT DELIVERY VEHICLE ROUTING PROBLEM ON A BIKE-SHARING SYSTEM

Over the past decade an increasing number of cities around the world have adopted bike-sharing systems. A bike-sharing system is a public service in which bicycles are made available for shared use to individuals on a short term basis. Typically, bikes are stored in rack stations. People rent a bike at a cost to travel around the city, and drop it back at either the same rack station or at a different one.

The Velib system in Paris, started in 2007, is a success story with more than 50 millions trips in its first two years of service [34]; in 2014 more than 800 cities across the globe had similar bike-sharing systems.

One of the main issues of bike-sharing systems is ensuring the availability of the bikes. In fact, during peak hours, flows along particular direction are registered, leading to high risk of empty racks in departure stations, and full racks at destination. Both represent a disservice, and may even prevent people to use the system, since the users are forced to spend time in searching for alternative stations in the neighbourhood.

One of the solutions chosen by many operators is to iteratively rebalance the system by means of a fleet of dedicated trucks: transportation demand is forecast, and bikes are picked up from stations where congestion is expected, and delivered to those expected to become empty. Due to the high costs of running trucks in a urban environment, efficient rebalancing operations are a key factor for the success of the whole system. Unfortunately, such operations require to solve very hard optimization problems.

In this thesis we face a *Split Pickup and Split Delivery Vehicle Routing Problem (SPSDVRP)* arising on such a bike-sharing system. We assume that it is given a homogeneous fleet of vehicles of limited capacity, a network of stations, the travel cost and time between them, a forecast of transportation demand and a current status of the network, expressed in terms of desired (resp. currently available) number of bikes at each station.

The SPSDVRP requires therefore to find a route for each vehicle, that is a pattern defining which stations need to be visited, the order

of visits, and the amount of bikes loaded or unloaded at each station. Due to capacity restrictions, no bikes can be loaded into a full vehicle, not unloaded from an empty one; similarly, no bike can be unloaded to a full station, and no more bikes can be loaded from a station than those actually available. Route length cannot exceed a given limit, representing the operator shift duration. We assume that each vehicles always starts and ends at a central depot with no bikes on board. We also assume that loading and unloading times are negligible with respect to travelling times. We finally assume that no station is used as a temporary unloading location; that is, in each station bikes can be only loaded or unloaded, and therefore during the balancing operations the number of bikes in each station is monotone. From a logistics point of view, without the latter assumption complex synchronization issues would arise, that would be very difficult to be implemented in practice. As a consequence, each station is classified since the beginning as either pickup, when more bikes are parked than the desired ones, or delivery, when instead more bikes are expected to be needed than the currently available ones.

A solution to the SPSDVRP consists of a set of routes respecting the above conditions, and such that the desired target demand is achieved for each station of the network. A solution is considered to be optimal when minimizing the sum of the travelling costs of all vehicles.

From an application point of view, there is currently a lively research trend in optimizing bike-sharing systems. In [52] the authors adopt a statistical approach to discuss the performances of existing systems. In [81], data from the Vienna bikes sharing system are gathered and studied to give a model that could be used to further expand the network. In [53] it is described a model that gives a strategic planning of a bike-sharing system by considering service level requirements. In [64] the authors propose several models and algorithms to solve bike repositioning problems. Their objective is to find the best repositioning that can be achieved by several vehicles within time limits in a static case, that is they assume a negligible usage rate of the system. The satisfaction function introduced in [63] is used to evaluate the quality of a repositioning. Both service level requirements and bike repositioning are combined in [67]. In [28] the authors propose to solve a dynamic public bike-sharing balancing problem. They introduce a time-discretized model of the system and use column generation techniques to obtain in short time instructions to be given to the drivers, in order to minimize the number of uncovered users.

From a methodological point of view, our SPSDVRP is NP-Hard, generalizing several problems in transportation. For instance, when all stations have a target number of bikes that is higher than the initial one, except for a single depot station, the SPSDVRP becomes a *Split Delivery Vehicle Routing Problem (SDVRP)* [5]. SDVRPs have been first studied in [38]. The authors proved that the split feature of the problem may lead to substantial savings, while increasing at the same time the complexity of the problem [5]. In [7] a tabu search algorithm to solve this problem is presented, while in [8] and [9] the authors propose exact algorithms exploiting column generation and cutting planes, respectively. In [35] a further degree of complexity is considered, forcing the deliveries to be satisfied within given time windows. The author proposes a branch-and-price approach to solve the problem to optimality.

The SPSDVRP belongs to the wide class of *Pickup and Delivery Vehicle Routing Problems* (PDVRPs), where a fleet of vehicles is used to transport supplies from either a depot or some selected vertices of the network, to either other vertices or back to the depot. For recent surveys on PDVRPs we refer to [12], covering freight transportation, and to [37] covering transportation of people. PDVRPs involving additional operational constraints have recently been addressed, as LIFO constraints on the loading/unloading [15], time windows [44], and both [24]. However, a substantial difference stands between a standard PDVRP and the SPSDVRP: while in the former the requests for each pair of pickup and delivery points are given, in the latter the quantity of supply transported from each pickup to each delivery point is a decision variable.

Finally, the SPSDVRP can be classified as a *many-to-many* (M-M) vehicle routing problem, in which a request has multiple origins (in our case pickup stations) and multiple destinations (in our case delivery stations). These kind of transportation problems arise for instance in maritime oil transportation [46].

The routing problem induced by balancing in bike-sharing systems with a single vehicle is addressed in [23], where the authors succeed in providing a strong lower bound and an effective heuristic. In [43] the authors propose a first branch-and-cut exploiting Benders' cuts. In [42] a variant of the problem is tackled, where the rebalancing requires to satisfy an interval demand; the authors exploit cutting planes methods to design exact algorithms. In [32], the authors propose new models and valid inequalities for the pure combinatorial version of

the problem, that is without considering the option of partially serving demands at each station.

A first mathematical programming algorithm for the SPSDVRP on a bike-sharing system has been proposed in [22]. The author models the problem by means of a set partitioning extended formulation in which each variable represents a full vehicle route, that is including its rebalancing pattern. First, the author obtains strong lower bounds by solving the continuous relaxation of his extended formulation by means of column generation techniques; to solve the pricing problem the author adapts an ad-hoc algorithm for the VRP first described in [11]. Secondly, he obtains tight upper bounds by means of a memetic algorithm. Third, he generates all columns with a reduced cost smaller than the gap between lower and upper bounds, following the technique of [11], and solves to integer optimality the resulting problem by means of general purpose integer programming solver.

The method of [22] has two main drawbacks: first, it is not designed to handle travelling times, that are instead approximated by a limit on the number of visits in each route; second, it is designed to tackle only instances with a very limited number of stations, due to the nature of its third step, and of the pricing problem to be solved during column generation.

We propose a new exact method for the SPSDVRP that overcomes these two drawbacks.

In Section 7.1 we formalize the problem; in Section 7.2 we propose a new mathematical programming model that makes use of combinations of suitable combinatorial structures to reduce the complexity of the problem. We then discuss about a few theoretical properties of such a model, and we propose an extended formulation obtained through Dantzig-Wolfe decomposition. In Section 7.3 we explain the details of our algorithm, while in Section 7.4 we show our computational results. Brief conclusions follow in Section 7.5.

MAIN CONTRIBUTIONS. In this part of the thesis we address the SPSDVRP on a bike-sharing system and improve the state of the art on such problem. In particular:

(a) we introduce a new formulation of the SPSDVRP in which routes are decomposed into simpler substructures, mitigating the combinatorial explosion of feasible solutions;

(b) we exploit properties on the structure of the solutions to improve the quality of the bound given by the continuous relaxation of our model;

(c) we obtain an extended formulation by means of Dantzig-Wolfe decomposition method and we make use of column generation techniques to solve its continuous relaxations, designing an ad-hoc algorithm for the pricing problem;

(d) we include the column generation procedure into a branch-and-price framework with new branching strategies and feasibility detection procedures;

(e) we study the performances of our algorithm by means of an extensive experimental campaign.

7.1 PROBLEM FORMALIZATION AND NOTATION

The SPSDVRP for a bike-sharing system can be formalized as follows: a set of station nodes $N = \{1 \dots n\}$ is given, each with an initial number $stock_i$ and a target number $target_i$ of bikes. When $stock_i > target_i$, i is defined as a *pickup* node, when $stock_i < target_i$ as a *delivery* node, and when $stock_i = target_i$ as a *balanced* node. Let us define $N^+ = \{i \in N \mid stock_i > target_i\}$ and $N^- = \{i \in N \mid stock_i < target_i\}$ as the set of pickup nodes and the set of delivery nodes, respectively. The demand of each node $d_i = |stock_i - target_i|$ is the quantity of bikes to pickup from (resp. deliver to) that node.

Let $G = (N_0, A)$ be a directed graph in which $N_0 = N \cup \{0\}$ is the set of nodes including the depot 0, and $A = \{(i,j) \mid i,j \in N_0\}$ is the set of arcs connecting them. Let c_{ij} be the travelling cost of arc $(i,j) \in A$. W.l.o.g we assume that travelling costs satisfy the triangular inequality, that is $c_{ij} \leqslant c_{ik} + c_{kj}$ for all $i,j,k \in N_0$. Let t_{ij} be the travelling time of arc $(i,j) \in A$; we assume that triangular inequalities hold also for travelling times, that is $t_{ij} \leqslant t_{ik} + t_{kj}$ for all $i,j,k \in N_0$.

An example of such an input of the SPSDVRP is shown in Figure 9; for clarity only a few arcs are depicted. In such figure, each node has two labels: one label that identifies the station i, and one attached label that is its demand d_i. Also, each node is denoted by a $^+$ or by a $^-$ if it is a pickup or a delivery node, respectively.

A homogeneous fleet of vehicles $M = \{1 \dots m\}$ each with capacity C is given to satisfy station node demands. Whenever a vehicle visits a station node, it may pick up or deliver a certain amount of bikes,

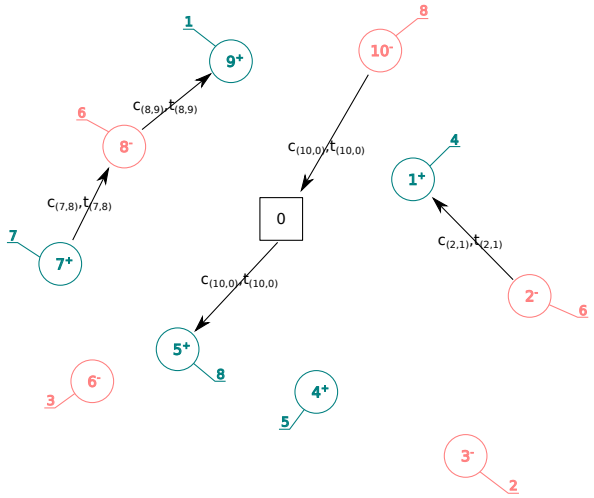

Figure 9: Example of instance with 10 station. Symbols $^+$ and $^-$ denote a pickup or a delivery station, respectively. At each station is attached a label reporting its demand.

depending on its current load. Since we assume that all vehicles begin and end their route empty, the sum of pickup demands must be equal to the sum of delivery demands, that is $\sum_{i \in N}(stock_i - target_i) = 0$ Moreover, each vehicle has a resource T that represent the available travelling time of such vehicle.

The SPSDVRP on a bike-sharing system is the problem of redistributing bikes in the network at minimum travelling cost, satisfying node demands while not exceeding neither vehicles capacity nor their time resource. In Figure 10 we depict an example in which we assume 2 vehicles with capacity $C = 10$ each. Both vehicles start empty from the depot and visit pickup node 5 splitting its demand. The load of vehicles after each operation is reported as a label on the arcs of the solutions. Also the demand of the delivery node 10 is split; then the two vehicles end their routes empty.

As discussed in the introduction, the convergence to the target state is required to be monotonous and drops are not allowed. It means that bikes can only be loaded at pickup vertices and unloaded at delivery vertices. As a consequence initially balanced vertices are not visited by any vehicle and so from now on we assume them to be removed from G. Instead, pickup and delivery vertices can be visited several times, either by the same vehicle or by different ones.

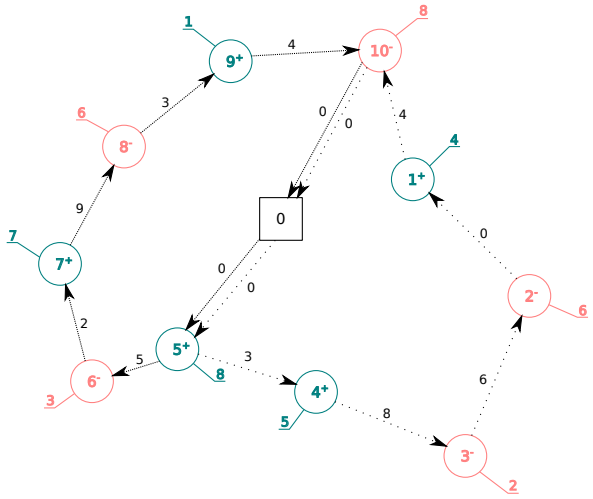

Figure 10: Example of feasible solution for the graph depicted in Figure 9 assuming a capacity C = 10 and using 2 vehicles. Each arc has a label reporting the load of the vehicle after visiting a node. Node 5 and node 10 are visited by both vehicles splitting their demands.

7.2 GROUPS FORMULATION AND PROPERTIES

The approach to the SPSDVRP proposed in [22], that modelled the problem as a set covering extended formulation in which each variable is a specific route pattern, revealed that solving the continuous relaxation of such formulation was very challenging due to the structure of the pricing problem. Indeed, we also performed preliminary solution attempts and experiments in that modelling direction; these however, confirmed the findings of [22].

That motivated us to elaborate on a different approach, identifying particular regularities and properties of combinatorial substructures of the routes, and trying to reduce the complexity of the pricing problem by exploiting these properties. Indeed, this kind of approach is in nature similar to those proposed in [59] and [21], that proved to be successful in similar contexts.

We first present some observations that led to our intuition (subsection 7.2.1), then we describe in detail our approach (subsections 7.2.1, 7.2.2, 7.2.3 and 7.2.4).

7.2.1 *Routes and groups*

We first observe that, due to triangular inequality:

Observation 7.2.1. *There always exists an optimal solution in which no node is visited without collecting at least one unit of its demand.*

Furthermore, since vehicles always start and end the route empty at the depot, we can observe that:

Observation 7.2.2. *There always exists an optimal solution in which no vehicle visits a delivery or a pickup station at the beginning or at the end of its route, respectively.*

These observations can be simply generalized as follows:

Observation 7.2.3. *A route always starts with a sequence composed only by pickup nodes, always ends with a sequence composed only by delivery nodes, and in general always interleaves sequences in which a set of pickup nodes are visited, without deliveries in between, followed by a set of visits to delivery nodes, without pickups in between.*

Our intuition is therefore that the structure of a route can be much simplified by explicitly encoding such an interleaved behaviour. Indeed, in the following we formalize such an intuition, and prove a few key properties of such an encoding.

Definition 7.2.1. *We denote as* group *a sequence of one or more pickup nodes followed by a sequence of one or more delivery nodes.*

Therefore, a route is itself a sequence of one or more groups linked together, plus an additional stop at the depot at the begin and at the end. An example of group structure in a route is depicted in Figure 11: the first route is partitioned in two groups, one with four and one with two nodes. The second route is partitioned in three groups with two nodes each. All groups start with a pickup node and end with a delivery node.

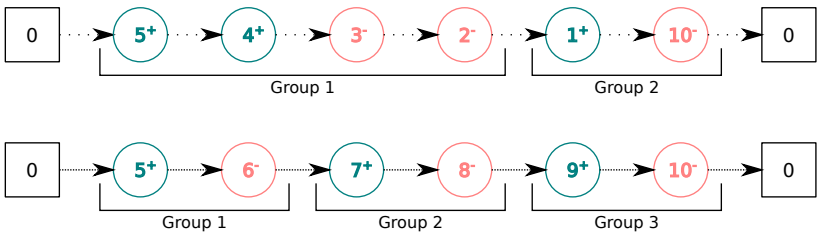

Figure 11: Group partitioning of routes in Figure 10

Definition 7.2.2. *A group is feasible if both the sum of loaded bikes and the sum of unloaded bikes do not exceed the capacity of the vehicle.*

Definition 7.2.3. *The cost of a group is given by the sum of the arcs connecting the nodes of the inner pickup and delivery sequences.*

Let g and g' be two consecutive groups of a route, and let i and j be the last and first node of group g and g', respectively. Then groups g and g' are connected in the route by an arc (i, j).

Definition 7.2.4. *The cost of a route is the sum of the cost of its groups and the cost of its connecting arcs.*

We readily observe that:

Observation 7.2.4. *There always exists an optimal solution in which no node is visited more than once in the same group.*

Proof. In fact, let us suppose by contradiction that such an optimal solution exists and that i is a pickup node visited twice in a group. Let q'_i and q''_i be the two quantities loaded on the vehicle, and that $\tilde{d}_i$ is the demand of i loaded, that is $\tilde{d}_i = q'_i + q''_i$. Since the visits occur in the same group, we know that between the first and the second visit there are only pickup nodes, and that loading $\tilde{d}_i$ units of demand does not exceed vehicle capacity. Therefore, we can set $q'_i = \tilde{d}_i$ and $q''_i = 0$, avoiding the second visit due to Observation 7.2.1. The same holds if i is a delivery node, visited more than once. □

We remark that such a property does not hold outside the group structure; that is, in general, visiting the same node twice may be both needed for feasibility or simply be profitable.

We also observe that:

Observation 7.2.5. *The number of bikes loaded in each node of a pickup sequence is irrelevant to both the feasibility and the cost of a group, as long as the total amount of loaded bikes remains constant. The same applies to the number of bikes loaded in each node of a delivery sequence.*

That is, each group can encode implicitly a potentially huge number of equivalent solutions.

7.2.2 *Routes, groups and loading patterns.*

We now consider the particular SPSDVRP subproblem arising when the nodes visiting sequence is assumed to be given, and only a suitable

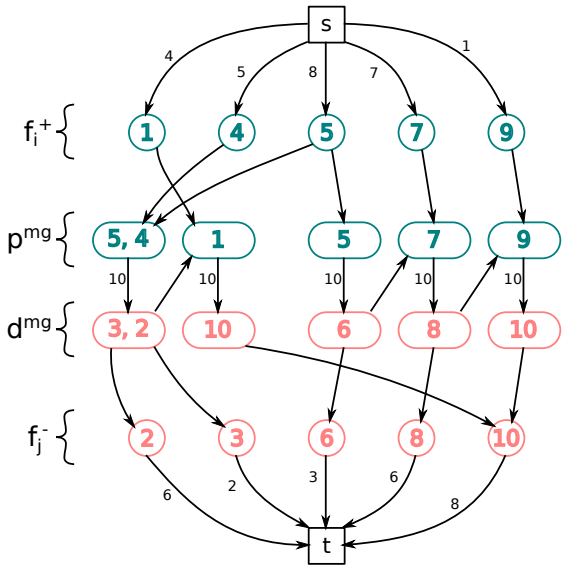

Figure 12: Resulting flow graph of the routes in the solution of Figure 10 assuming vehicles with capacity $C = 10$. The labels on the arcs are the capacities of each arc (infinite capacities are omitted).

loading/unloading plan needs to be found. Our main result is that by exploiting the groups structure we are able to improve theoretical findings from the literature.

Theorem 7.2.1. *Given the sets of nodes visited in each group of each vehicle, the problem of assigning the maximum quantity loaded (resp. unloaded) at each station can be solved in polynomial time.*

Proof. Let us build a graph in which we have a source node s and a sink node t, a node f_i^+ for each pickup node i and a node f_j^- for each delivery node j, and two nodes p^{mg} and d^{mg} for each group g of each vehicle m. Let us add arcs from s to f_i^+ and from f_j^- to t with capacity d_i and d_j, respectively. For each pickup node i visited in a group g of vehicle m, add an arc from f_i^+ to p^{mg} with infinite capacity. Similarly, for each delivery node j visited in a group g of vehicle m, add an arc from d^{mg} to f_j^- still with infinite capacity. From each node p^{mg} add an arc to d^{mg} with capacity C, and from each node d^{mg} add an arc to p^{mg+1} with infinite capacity. An example of the graph is reported in Figure 12.

A maximum flow solution on such a graph ensures that at most $\sum_{i \in N^+} d_i$ units can be loaded, and at most $\sum_{i \in N^-} d_i$ units unloaded. For each station node, the units loaded (unloaded) are at most the demand of the station itself because of the limited capacity of ingoing

arcs in f_i^+ (outgoing arcs from f_j^-). Nodes p^{mg} and d^{mg} represent the load of the vehicle after pickups and deliveries, respectively. No vehicle is overloaded due to the limited capacity of arcs (p^{mg}, d^{mg}).

The quantities loaded and unloaded at nodes i and j of a group g of vehicle m are given by the flow on the arc (f_i^+, p^{mg}) and (d^{mg}, f_j^-), respectively. □

Corollary 7.2.2. *If the flow reaching the depot t is less than the number of demands of pickup (delivery) nodes, then the starting assignment of nodes to groups does not represent a feasible solution.*

This follows from the fact that some demands are not satisfied if the flow is less then their sum.

Theorem 7.2.1 and Corollary 7.2.2 imply:

Observation 7.2.6. *Given a set of routes without loading (unloading) information, it is always possible to complete the solution with such quantities in polynomial time, or prove that such solution is infeasible.*

Our approach is indeed similar to the one presented in [22], where given a set of routes without loading quantities, the authors use a flow formulation to obtain such a missing information. Indeed the claim of our theorem 7.2.1 matches the findings of [22].

However, our proof is different: from a theoretical point of view, our approach allows to build smaller support graph, and may therefore yield a better computational behaviour. In details, the graph of [22] has two nodes for each station of the problem, while our graph has one node for each station, and two nodes for each group. Therefore, our graph has always a smaller number of nodes, except in the extreme scenario in which exactly one pickup and one delivery node is visited in each group. In such scenario, the number of nodes of the two graphs is identical. Furthermore, our approach requires less information about the order of the nodes in the route, and can be used to early detect the infeasibility of a node in a branch-and-bound approach as described in Subsection 7.3.6.

7.2.3 *A formulation based on groups*

Then we exploit the features of groups to obtain a new formulation of the SPSDVRP. To ease notation we assume that there exists a particular additional group containing the depot only. Let $G = \{1 \dots g\}$ be the set

of available groups for each vehicle, the SPSDVRP can be formulated as follows:

$$\min \quad \sum_{\substack{m\in M\\ g\in G}} \sum_{\substack{i\in N_0^-\\ j\in N_0^+}} c_{ij}\cdot x_{ij}^{mg} + \sum_{i,j\in N} c_{ij}\cdot z_{ij}^{mg} \tag{7.2.1}$$

$$\text{s.t.} \quad \sum_{\substack{m\in M\\ g\in G}} w_i^{mg} \geqslant 1 \qquad \forall i\in N \tag{7.2.2}$$

$$\sum_{\substack{m\in M\\ g\in G}} q_i^{mg} = d_i \qquad \forall i\in N \tag{7.2.3}$$

$$q_i^{mg} \leqslant d_i\cdot w_i^{mg} \qquad \substack{\forall m\in M\\ \forall g\in G\\ \forall i\in N} \tag{7.2.4}$$

$$q_i^{mg} \geqslant w_i^{mg} \qquad \substack{\forall m\in M\\ \forall g\in G\\ \forall i\in N} \tag{7.2.5}$$

$$\sum_{i\in N^+} q_i^{mg} \leqslant C \qquad \substack{\forall m\in M\\ \forall g\in G} \tag{7.2.6}$$

$$\sum_{i\in N^-} q_i^{mg} \leqslant C \qquad \substack{\forall m\in M\\ \forall g\in G} \tag{7.2.7}$$

$$f^{mg} + \sum_{i\in N^+} q_i^{mg+1} - \sum_{i\in N^-} q_i^{mg+1} = f^{mg+1} \qquad \substack{\forall m\in M\\ \forall g\in G} \tag{7.2.8}$$

$$f^{mg} + \sum_{i\in N^+} q_i^{mg+1} \leqslant C\cdot(1-w_0^{mg}) \qquad \substack{\forall m\in M\\ \forall g\in G} \tag{7.2.9}$$

$$\sum_{i\in N} w_i^{mg} \leqslant |N|\cdot(1-w_0^{mg}) \qquad \substack{\forall m\in M\\ \forall g\in G} \tag{7.2.10}$$

$$\sum_{j\in N} z_{ij}^{mg} = \sum_{j\in N^+} z_{ji}^{mg} + s_i^{mg} = w_i^{mg} \qquad \substack{\forall m\in M\\ \forall g\in G\\ \forall i\in N^+} \tag{7.2.11}$$

$$\sum_{j\in N^-} z_{ij}^{mg} + e_i^{mg} = \sum_{j\in N} z_{ji}^{mg} = w_i^{mg} \qquad \substack{\forall m\in M\\ \forall g\in G\\ \forall i\in N^-} \tag{7.2.12}$$

$$1 \geqslant \sum_{\substack{i\in N^+\\ j\in N^-}} z_{ij}^{mg} \geqslant w_u^{mg} \qquad \substack{\forall m\in M\\ \forall g\in G\\ u\in N} \tag{7.2.13}$$

$$\sum_{\substack{i\in N\setminus S\\ j\in S}} (z_{ij}^{mg} + z_{ji}^{mg}) \geqslant w_u^{mg} \qquad \substack{m\in M\\ g\in G\\ S\subset N\\ |S|>0\\ u\in S} \tag{7.2.14}$$

$$\sum_{i\in N_0^-} e_i^{mg} \leqslant 1 \qquad \substack{\forall m\in M\\ \forall g\in G} \tag{7.2.15}$$

$$\sum_{i\in N_0^+} s_i^{mg} \leqslant 1 \qquad \substack{\forall m\in M\\ \forall g\in G} \tag{7.2.16}$$

$$w_0^{mg} \geqslant e_0^{mg} + s_0^{mg} \qquad \substack{\forall m\in M\\ \forall g\in G} \tag{7.2.17}$$

$$e_i^{mg} \leqslant \sum_{j\in N_0^+} x_{ij}^{mg} \qquad \substack{\forall m\in M\\ \forall g\in G\\ i\in N_0^-} \tag{7.2.18}$$

$$s_i^{mg+1} \leqslant \sum_{j \in N_0^-} x_{ji}^{mg} \qquad \forall m \in M,\ \forall g \in G,\ i \in N_0^+ \qquad (7.2.19)$$

$$e_0^{m1} = \sum_{\substack{g \in G \\ g>1}} s_0^{mg} = 1 \qquad \forall m \in M \qquad (7.2.20)$$

$$\sum_{i,j \in N} x_{ij}^{mg} \leqslant \sum_{i,j \in N_0} x_{ij}^{mg-1} - \sum_{i \in N} x_{i0}^{mg-1} \qquad \forall m \in M,\ \forall g \in G \qquad (7.2.21)$$

$$\sum_{\substack{i \in N_0^- \\ j \in N_0^+}} x_{ij}^{mg} \leqslant 1 \qquad \forall m \in M,\ \forall g \in G \qquad (7.2.22)$$

$$\sum_{\substack{i \in N_0^- \\ j \in N_0^+}} t_{ij} \cdot x_{ij}^{mg} + \sum_{i,j \in N} t_{ij} \cdot z_{ij}^{mg} \leqslant T \qquad \forall m \in M,\ \forall g \in G \qquad (7.2.23)$$

$$x_{ij}^{mg} \in \mathbb{B} \qquad \forall m \in M,\ \forall g \in G,\ \forall i \in N_0^-,\ j \in N_0^+ \qquad (7.2.24)$$

$$z_{ij}^{mg} \in \mathbb{B} \qquad \forall m \in M,\ \forall g \in G,\ \forall i,j \in N \qquad (7.2.25)$$

$$w_i^{mg} \in \mathbb{B} \qquad \forall m \in M,\ \forall g \in G,\ \forall i \in N \qquad (7.2.26)$$

$$0 \leqslant f^{mg} \leqslant C \qquad \forall m \in M,\ \forall g \in G \qquad (7.2.27)$$

$$q_i^{mg} \in \mathbb{N}_0 \qquad \forall m \in M,\ \forall g \in G,\ \forall i \in N \qquad (7.2.28)$$

$$s_i^{mg} \in \mathbb{B} \qquad \forall m \in M,\ \forall g \in G,\ \forall i \in N_0 \qquad (7.2.29)$$

$$e_i^{mg} \in \mathbb{B} \qquad \forall m \in M,\ \forall g \in G,\ \forall i \in N_0 \qquad (7.2.30)$$

Variables x_{ij}^{mg} and z_{ij}^{mg} correspond respectively to the linking arcs between groups and the linking arcs inside a group; the former are set to 1 if there is an arc between group g and group $g+1$ that connect node i and j, while the latter are set to 1 if there is an arc between node i and node j inside the group. Variables w_i^{mg} are set to 1 if node i is visited by vehicle m in group g. Variables s_i^{mg} and e_i^{mg} are set to 1 if i is the starting node of the group g or if i is the ending node respectively. Variable q_i^{mg} is the quantity loaded (unloaded) at node i by vehicle m in group g, while variable f^{mg} is the load of the vehicle m after visiting group g.

The objective function (7.2.1) minimizes the overall cost by minimizing both group costs and linking arc costs. Constraints (7.2.2) and (7.2.3) ensure respectively that each station is visited at least once and its demand is satisfied. Constraints (7.2.4) avoid loading (unloading) when a node is not visited, while constraints (7.2.5) impose that if

a node is visited, then at least one unit of demand is loaded. Constraints (7.2.6) and (7.2.7) ensure that vehicle capacity is not exceeded for each group. Constraints (7.2.8) ensure consistency of the flow in the route, while constraints (7.2.9) impose that the vehicle is empty when visiting depot and therefore a vehicle starts and ends empty. Constraints (7.2.10) impose that no station is visited in a group if the depot is visited, too. Constraints (7.2.11) and (7.2.12) ensure respectively that all pickup and delivery nodes visited have ingoing and outgoing arcs. Constraints (7.2.13) ensure that in each group with a pickup or delivery node, there is an arc going from pickup to delivery, while constraints (7.2.14) ensure that there are no subtours in a group. Constraints (7.2.15) and (7.2.16) impose that for each group there must be at most one ending node and one starting node, while constraints (7.2.17) impose that if a vehicle visits the depot, it is only to start or to end a route. Constraints (7.2.18) and (7.2.19) impose the use of linking arcs between each group, while constraints (7.2.20) has the double effect of ensure that each vehicle visits the depot twice, and that its route starts from the depot. Constraints (7.2.21) ensure that no arcs are used after the depot is visited and constraints (7.2.22) impose that at most one linking arc is used for each group. Finally, constraints (7.2.23) impose a limit on the time resource consumed by each vehicle.

AN UPPER BOUND ON THE NUMBER OF GROUPS. In model (7.2.1) – (7.2.30), the total number of groups is not known in advance. However, since a maximal resource T is given, we observe that:

Observation 7.2.7. *An upper bound n_{max} on the maximal number of nodes visited in a route can be obtained by solving a* 0-1 *Knapsack Problem (KP).*

In fact, we can model the problem of finding the maximal number of nodes visited in a single route as follows:

$$\max \quad \sum_{i \in N_0} w_i \tag{7.2.31}$$

$$\text{s.t.} \quad \sum_{i \in N} w_i \cdot \left(\min_{j \in N_0} t_{ji} \right) \leqslant T \tag{7.2.32}$$

$$w_0 = 1 \tag{7.2.33}$$

$$w_i \in \mathbb{B} \qquad \forall i \in N \tag{7.2.34}$$

where w_i is set to 1 if node i is visited.

The objective function (7.2.31) maximizes the number of nodes visited. Constraint (7.2.32) ensures that resource T is not exceeded. Constraint (7.2.33) imposes that the depot is always visited.

Problem (7.2.31) – (7.2.34) can be solved as a KP in which (a) the vector of prizes is $(1 \dots 1)$, (b) each item weight is the minimum ingoing (outgoing) arc, and (c) node 0 is always included, therefore decreasing the resource T by $\min_{j \in N_0} t_{j0}$. Such a problem can be solved in polynomial time by packing items with smallest $\min_{j \in N_0} t_{ji}$ first.

Furthermore, the maximum number of visited nodes directly influences the maximum number of groups of each vehicle:

Observation 7.2.8. *For each route of a vehicle* m*, there are at most*

$$g^m_{max} = \left\lfloor \frac{n_{max} - 1}{2} \right\rfloor + 2$$

groups.

Given a fixed number of nodes, maximizing the number of groups is equivalent to minimize the number of nodes in each group. Therefore, we obtain the maximal number of groups when there are at most two nodes per group, one pickup node and one delivery node. Since the depot is always visited twice, once at the being and once at the end of the route, we need two additional groups.

7.2.4 *Extended formulation*

In order to obtain tight dual bounds to be used in search tree algorithms, we built an extended formulation of the model (7.2.1) – (7.2.30) exploiting Dantzig-Wolfe decomposition [30]. Let, for each vehicle $m \in M$ and group $g \in G$,

$$\Omega_{mg} = \left\{ (z, w, q, s, e) \in \mathbb{B}^{|N| \cdot |N|} \times \mathbb{B}^{|N_0|} \times \mathbb{N}_0^{|N|} \times \mathbb{B}^{|N_0^+|} \times \mathbb{B}^{|N_0^-|} \right\}$$

be a set of feasible integer points, where each vector (z, w, q, s, e) satisfies the constraints

$$
\begin{aligned}
& q_i \leqslant d_i \cdot w_i && \forall i \in N \\
& q_i \geqslant w_i && \forall i \in N \\
& \sum_{i \in N^+} q_i \leqslant C \\
& \sum_{i \in N^-} q_i \leqslant C \\
& \sum_{i \in N} w_i \leqslant |N| \cdot (1 - w_0) \\
& \sum_{j \in N} z_{ij} = \sum_{j \in N^+} z_{ji} + s_i = w_i && \forall i \in N^+ \\
& \sum_{j \in N^-} z_{ij} + e_i = \sum_{j \in N} z_{ji} = w_i && \forall i \in N^- \\
& 1 \geqslant \sum_{\substack{i \in N^+ \\ j \in N^-}} z_{ij} \geqslant w_u && u \in N \\
& \sum_{\substack{i \in N \setminus S \\ j \in S}} (z_{ij} + z_{ji}) \geqslant w_u && \substack{S \subset N \\ |S| > 0 \\ u \in S} \\
& \sum_{i \in N_0^-} e_i \leqslant 1 \\
& \sum_{i \in N_0^+} s_i \leqslant 1 \\
& w_0 \geqslant e_0 + s_0.
\end{aligned}
$$

We relax integrality conditions, but replace each Ω^{mg} with the convex hull of its L_{mg} extreme integer points

$$\Gamma_{mg} = \left\{ (\bar{z}^1, \bar{w}^1, \bar{q}^1, \bar{s}^1, \bar{e}^1), \ldots, (\bar{z}^{L_{mg}}, \bar{w}^{L_{mg}}, \bar{q}^{L_{mg}}, \bar{s}^{L_{mg}}, \bar{e}^{L_{mg}}) \right\}$$

and we impose

$$(z, w, q, s, e) = \sum_{k \in \Gamma_{mg}} (\bar{z}^k, \bar{w}^k, \bar{q}^k, \bar{s}^k, \bar{e}^k) \cdot y^k \tag{7.2.35}$$

with $y^k \geqslant 0$ for each $k \in \Gamma_{mg}$, $m \in M$, and $g \in G$, and $\sum_{k \in \Gamma_{mg}} y^k = 1$ for each $m \in M$ and $g \in G$. That is, each point is represented as a linear convex combination of points in Γ_{mg}.

The model obtained by replacing in the continuous relaxations of formulation (7.2.1) – (7.2.30) the vectors (z, w, q, s, e) as indicated in (7.2.35), and by making explicit the vector indices is

$$\min \quad \sum_{\substack{m\in M\\ g\in G}} \sum_{\substack{i\in N_0^-\\ j\in N_0^+}} c_{ij}\cdot x_{ij}^{mg} + \sum_{\substack{i,j\in N\\ k\in \Gamma_{mg}}} c_{ij}\cdot \bar{z}_{ij}^{k}\cdot y^{k} \tag{7.2.36}$$

$$\text{s.t.} \quad \sum_{\substack{m\in M\\ g\in G\\ k\in \Gamma_{mg}}} \bar{w}_i^k\cdot y^k \geqslant 1 \qquad \forall i\in N \tag{7.2.37}$$

$$\sum_{\substack{m\in M\\ g\in G\\ k\in \Gamma_{mg}}} \bar{q}_i^k\cdot y^k = d_i \qquad \forall i\in N \tag{7.2.38}$$

$$f^{mg} + \sum_{\substack{i\in N^+\\ k\in \Gamma_{mg+1}}} \bar{q}_i^k\cdot y^k - \sum_{\substack{i\in N^-\\ k\in \Gamma_{mg+1}}} \bar{q}_i^k\cdot y^k = f^{mg+1} \qquad \forall m\in M,\ \forall g\in G \tag{7.2.39}$$

$$f^{mg} + \sum_{\substack{i\in N^+\\ k\in \Gamma_{mg+1}}} \bar{q}_i^k\cdot y^k \leqslant C\cdot\Big(1 - \sum_{k\in\Gamma_{mg}} \bar{w}_0^k\cdot y^k\Big) \qquad \forall m\in M,\ \forall g\in G \tag{7.2.40}$$

$$\sum_{k\in\Gamma_{mg}} \bar{e}_i^k\cdot y^k \leqslant \sum_{j\in N_0^+} x_{ij}^{mg} \qquad \forall m\in M,\ \forall g\in G,\ i\in N_0^- \tag{7.2.41}$$

$$\sum_{k\in\Gamma_{mg+1}} \bar{s}_i^k\cdot y^k \leqslant \sum_{j\in N_0^-} x_{ji}^{mg} \qquad \forall m\in M,\ \forall g\in G,\ i\in N_0^+ \tag{7.2.42}$$

$$\sum_{k\in\Gamma_{mg}} \bar{e}_0^{m1}\cdot y^k = \sum_{\substack{g\in G\\ g>1\\ k\in\Gamma_{mg}}} \bar{s}_0^{mg}\cdot y^k = 1 \qquad \forall m\in M \tag{7.2.43}$$

$$\sum_{\substack{i\in N_0^-\\ j\in N_0^+}} t_{ij}\cdot x_{ij}^{mg} + \sum_{\substack{i,j\in N\\ k\in\Gamma_{mg}}} t_{ij}\cdot \bar{z}_{ij}^{k}\cdot y^k \leqslant T \qquad \forall m\in M,\ \forall g\in G \tag{7.2.44}$$

$$\sum_{i,j\in N} x_{ij}^{mg} \leqslant \sum_{i,j\in N_0} x_{ij}^{mg-1} - \sum_{i\in N} x_{i0}^{mg-1} \qquad \forall m\in M,\ \forall g\in G \tag{7.2.45}$$

$$\sum_{\substack{i\in N_0^-\\ j\in N_0^+}} x_{ij}^{mg} \leqslant 1 \qquad \forall m\in M,\ \forall g\in G \tag{7.2.46}$$

$$\sum_{k\in\Gamma_{mg}} y^k = 1 \qquad \forall m\in M,\ \forall g\in G \tag{7.2.47}$$

$$y^k \geqslant 0 \qquad \forall m\in M,\ \forall g\in G,\ k\in\Gamma_{mg} \tag{7.2.48}$$

$$0 \leqslant x_{ij}^{mg} \leqslant 1 \qquad \forall m\in M,\ \forall g\in G,\ \forall i\in N_0^-,\ j\in N_0^+ \tag{7.2.49}$$

$$0 \leqslant f^{mg} \leqslant C \qquad \forall m\in M,\ \forall g\in G \tag{7.2.50}$$

Constraints (7.2.47) may be first relaxed in $\leqslant$ form, because a pattern $(0 \dots 0)$ always exists for each set Γ_{mg}, and then can be removed because constraints (7.2.46) already impose at most one linking arc and therefore at most one selected group.

Observation 7.2.9. *The lower bound provided by the (7.2.36) – (7.2.50) is at least as tight as that given by the continuous relaxation of model (7.2.1) – (7.2.30).*

To strengthen the formulation and reduce symmetries we add for each vehicle $m \in M$ and consecutive groups g and $g+1$ the inequality

$$\sum_{k \in \Gamma_{mg+1}} y^k = \sum_{k \in \Gamma_{mg}} y^k - \sum_{k \in \Gamma_{mg}} w_0^k \cdot y^k, \tag{7.2.51}$$

and for each consecutive vehicles m and $m+1$ and group $g \in G$, the inequality

$$\sum_{i \in N_0^-, j \in N_0^+} x_{ij}^{mg} \geqslant \sum_{i \in N_0^-, j \in N_0^+} x_{ij}^{m+1g} \tag{7.2.52}$$

The model can then be rewritten as

$$\min \sum_{\substack{m \in M \\ g \in G}} \sum_{\substack{i \in N_0^- \\ j \in N_0^+}} c_{ij} \cdot x_{ij}^{mg} + \sum_{\substack{i,j \in N \\ k \in \Gamma_{mg}}} c_{ij} \cdot \bar{z}_{ij}^k \cdot y^k \tag{7.2.53}$$

$$\text{s.t.} \sum_{\substack{m \in M \\ g \in G \\ k \in \Gamma_{mg}}} \bar{w}_i^k \cdot y^k \geqslant 1 \qquad \forall i \in N \tag{7.2.54}$$

$$\sum_{\substack{m \in M \\ g \in G \\ k \in \Gamma_{mg}}} \bar{q}_i^k \cdot y^k = d_i \qquad \forall i \in N \tag{7.2.55}$$

$$f^{mg} + \sum_{\substack{i \in N^+ \\ k \in \Gamma_{mg+1}}} \bar{q}_i^k \cdot y^k - \sum_{\substack{i \in N^- \\ k \in \Gamma_{mg+1}}} \bar{q}_i^k \cdot y^k = f^{mg+1} \qquad \begin{matrix} \forall m \in M \\ \forall g \in G \end{matrix} \tag{7.2.56}$$

$$f^{mg} + \sum_{\substack{i \in N^+ \\ k \in \Gamma_{mg+1}}} \bar{q}_i^k \cdot y^k + C \cdot \sum_{k \in \Gamma_{mg}} \bar{w}_0^k \cdot y^k \leqslant C \qquad \begin{matrix} \forall m \in M \\ \forall g \in G \end{matrix} \tag{7.2.57}$$

$$\sum_{k \in \Gamma_{mg}} \bar{e}_i^k \cdot y^k - \sum_{j \in N_0^+} x_{ij}^{mg} \leqslant 0 \qquad \begin{matrix} \forall m \in M \\ \forall g \in G \\ i \in N_0^- \end{matrix} \tag{7.2.58}$$

$$\sum_{k \in \Gamma_{mg+1}} \bar{s}_i^k \cdot y^k - \sum_{j \in N_0^-} x_{ji}^{mg} \leqslant 0 \qquad \begin{matrix} \forall m \in M \\ \forall g \in G \\ i \in N_0^+ \end{matrix} \tag{7.2.59}$$

$$\sum_{k\in\Gamma_{mg}} \bar{e}_0^{m1}\cdot y^k - \sum_{\substack{g\in G\\ g>1\\ k\in\Gamma_{mg}}} \bar{s}_0^{mg}\cdot y^k = 0 \qquad \forall m\in M \qquad (7.2.60)$$

$$\sum_{k\in\Gamma_{mg}} \bar{e}_0^{m1}\cdot y^k = 1 \qquad \forall m\in M \qquad (7.2.61)$$

$$\sum_{\substack{i\in N_0^-\\ j\in N_0^+}} t_{ij}\cdot x_{ij}^{mg} + \sum_{\substack{i,j\in N\\ k\in\Gamma_{mg}}} t_{ij}\cdot \bar{z}_{ij}^k\cdot y^k \leqslant T \qquad \begin{matrix}\forall m\in M\\ \forall g\in G\end{matrix} \qquad (7.2.62)$$

$$\sum_{i,j\in N} x_{ij}^{mg} - \sum_{i,j\in N_0} x_{ij}^{mg-1} + \sum_{i\in N} x_{i0}^{mg-1} \leqslant 0 \qquad \begin{matrix}\forall m\in M\\ \forall g\in G\end{matrix} \qquad (7.2.63)$$

$$\sum_{\substack{i\in N_0^-\\ j\in N_0^+}} x_{ij}^{mg} \leqslant 1 \qquad \begin{matrix}\forall m\in M\\ \forall g\in G\end{matrix} \qquad (7.2.64)$$

$$\sum_{k\in\Gamma_{mg+1}} y^k - \sum_{k\in\Gamma_{mg}} y^k + \sum_{k\in\Gamma_{mg}} w_0^k\cdot y^k = 0 \qquad \begin{matrix}\forall m\in M\\ \forall g\in G\end{matrix} \qquad (7.2.65)$$

$$\sum_{\substack{i\in N_0^-\\ j\in N_0^+}} x_{ij}^{mg} - \sum_{\substack{i\in N_0^-\\ j\in N_0^+}} x_{ij}^{m+1g} \geqslant 0 \qquad \begin{matrix}\forall m\in M\\ \forall g\in G\end{matrix} \qquad (7.2.66)$$

$$y^k \geqslant 0 \qquad \begin{matrix}\forall m\in M\\ \forall g\in G\\ k\in\Gamma_{mg}\end{matrix} \qquad (7.2.67)$$

$$0 \leqslant x_{ij}^{mg} \leqslant 1 \qquad \begin{matrix}\forall m\in M\\ \forall g\in G\\ \forall i\in N_0^-\\ j\in N_0^+\end{matrix} \qquad (7.2.68)$$

$$0 \leqslant f^{mg} \leqslant C \qquad \begin{matrix}\forall m\in M\\ \forall g\in G\end{matrix} \qquad (7.2.69)$$

7.3 ALGORITHMS

The size of the set Γ grows exponentially in the number of nodes and the sum of demands; therefore, we solve the MP by means of column generation techniques: we solve to optimality a Restricted Master Problem (RMP) involving a small set of columns (see Subsection 7.3.1), and we iteratively search for negative reduced cost variables solving a pricing problem (see Subsection 7.3.2). If no negative reduced cost variable is found, the optimal RMP solution is optimal for the MP as well, and therefore the corresponding value is retained as a valid lower bound for the SPSDVRP. If the final RMP solution is integer, then it is also optimal for the SPSDVRP, otherwise we enter a search tree by performing branching operations (see Subsection 7.3.3) to find a proven global optimum.

7.3.1 *Initialization*

We initialize the RMP by generating the set of columns corresponding to groups including the depot only. We remark that if a group contains the depot, then it does not contain any station. Our aim is to both simplify the pricing problem by removing a decision variable and to populate the RMP with a set of columns that are always needed to obtain a feasible solution. Therefore, for each vehicle we add an initial depot in group 1, and an ending depot in all groups $g > 1$.

In model (7.2.53) – (7.2.69), constraints (7.2.63) impose that a vehicle cannot visit a station after the ending depot. Indeed, if a vehicle m visits the depot in group 2, that is

$$\sum_{k \in \Gamma_{m2}} \bar{w}_0^k \cdot y^k = 1,$$

then m visits the depot in both groups 1 and 2, and therefore it does not visits any station.

For a few vehicles we can forbid such a redundant behaviour. Let us consider for example vehicle 1: due to symmetry constraints (7.2.66), if vehicle 1 visits the depot in group 2, then no other vehicle can leave the depot nor visit any station.

Observation 7.3.1. *Let g_{min} be the minimum number of groups required to perform all the pickups and deliveries, that is*

$$g_{min} = \left\lceil \frac{\sum_{i \in N} d_i}{2} \right\rceil,$$

we avoid columns with depot only in group 2 of vehicle m if

$$(m-1) \cdot (g_{max}^m - 2) < g_{min}.$$

In other words, a vehicle may be left unused only if the previous ones are able to perform all the pickups and deliveries.

7.3.2 *Pricing problem*

Due to our initialization method, the pricer can neglect groups including the depot.

Let π, λ, μ, ζ, ν^+, ν^-, η, and θ be respectively the dual variables of constraints (7.2.54), (7.2.55), (7.2.56), (7.2.57), (7.2.58), (7.2.59), (7.2.62),

and (7.2.65). Since ν^+ and ν^- are referred to the two sets of nodes N^+ and N^-, for each node $i \in N$ we use instead

$$\nu_i = \begin{cases} \nu_i^+ & \text{if } i \in N^+ \\ \nu_i^- & \text{if } i \in N^- \end{cases}.$$

For each $k \in \Gamma_{mg}$, the reduced cost of variable y^k is computed as

$$\begin{aligned}\sigma^k = &\sum_{i,j\in N} c_{ij} \cdot \bar{z}_{ij}^k - \sum_{i,j\in N} \eta^{mg} \cdot t_{ij} \cdot \bar{z}_{ij}^k - \sum_{i\in N} \pi_i \cdot \bar{w}_i^k \\ &- \sum_{i\in N} \lambda_i \cdot \bar{q}_i^k - \sum_{i\in N^+} \mu^{mg} \cdot \bar{q}_i^k + \sum_{i\in N^-} \mu^{mg} \cdot \bar{q}_i^k - \sum_{i\in N^+} \zeta^{mg} \cdot \bar{q}_i^k \\ &- \sum_{i\in N^+} \nu_i^{mg} \cdot \bar{s}_i^k - \sum_{i\in N^-} \nu_i^{mg} \cdot \bar{e}_i^k - \theta^{mg} + \theta^{mg+1}.\end{aligned}$$

The component $-\theta^{mg} + \theta^{mg+1}$ is a fixed prize (cost) gained (paid) for any additional column and thus can be ignored during pricing problem optimization. After collecting the coefficients, the pricing objective function reads as follows:

$$\begin{aligned}\sigma^k = &\sum_{i,j\in N} (c_{ij} - \eta^{mg} \cdot t_{ij}) \cdot \bar{z}_{ij}^k - \sum_{i\in N} \pi_i \cdot \bar{w}_i^k \\ &- \sum_{i\in N^+} (\lambda_i + \mu^{mg} + \zeta^{mg}) \cdot \bar{q}_i^k - \sum_{i\in N^-} (\lambda_i - \mu^{mg}) \cdot \bar{q}_i^k \\ &- \sum_{i\in N^+} \nu_i^{mg} \cdot \bar{s}_i^k - \sum_{i\in N^-} \nu_i^{mg} \cdot \bar{e}_i^k\end{aligned}$$

Let

- $\alpha_{ij}^{mg} = c_{ij} - \eta^{mg} \cdot t_{ij}$ be the cost of travelling on arc (i, j);
- $\beta_i^{mg} = \lambda_i + \mu^{mg} + \zeta^{mg}$ be the prize or cost of loading one unit in node $i \in N^+$ and $\beta_i^{mg} = \lambda_i - \mu^{mg}$ be the prize or cost of loading one unit of node $i \in N^-$;
- γ_i^{mg} be the cost of starting the group visiting node $i \in N^+$ and γ_i^{mg} be the cost of ending the group visiting node $i \in N^-$.

The objective function can be now stated as

$$\begin{aligned}\sigma^k = &\sum_{i,j\in N} \alpha_{ij}^{mg} \cdot \bar{z}_{ij}^k - \sum_{i\in N} \pi_i \cdot \bar{w}_i^k - \sum_{i\in N} \beta_i^{mg} \cdot \bar{q}_i^k \\ &- \sum_{i\in N^+} \gamma_i^{mg} \cdot \bar{s}_i^k - \sum_{i\in N^-} \gamma_i^{mg} \cdot \bar{e}_i^k.\end{aligned}$$

For each vehicle $m \in M$ and group $g \in G$, we can formulate the pricing problem as follows:

$$\begin{aligned}
\min \quad & \sum_{i,j\in N} \alpha_{ij}^{mg} \cdot \bar{z}_{ij}^{k} - \sum_{i\in N} \pi_i \cdot \bar{w}_i^k - \sum_{i\in N} \beta_i^{mg} \cdot \bar{q}_i^k \\
& - \sum_{i\in N^+} \gamma_i^{mg} \cdot \bar{s}_i^k - \sum_{i\in N^-} \gamma_i^{mg} \cdot \bar{e}_i^k && (7.3.1)\\
\text{s.t.} \quad & \bar{q}_i^k \leqslant d_i \cdot \bar{w}_i^k && \forall i\in N \quad (7.3.2)\\
& \bar{q}_i^k \geqslant w_i^k && \forall i\in N \quad (7.3.3)\\
& \sum_{i\in N^+} \bar{q}_i^k \leqslant C && (7.3.4)\\
& \sum_{i\in N^-} \bar{q}_i^k \leqslant C && (7.3.5)\\
& \sum_{j\in N} \bar{z}_{ij}^k = \sum_{j\in N^+} \bar{z}_{ji}^k + \bar{s}_i^k = \bar{w}_i^k && \forall i\in N^+ \quad (7.3.6)\\
& \sum_{j\in N^-} \bar{z}_{ij}^k + \bar{e}_i^k = \sum_{j\in N} \bar{z}_{ji}^k = \bar{w}_i^k && \forall i\in N^- \quad (7.3.7)\\
& 1 \geqslant \sum_{\substack{i\in N^+\\ j\in N^-}} \bar{z}_{ij}^k \geqslant \bar{w}_u^k && u\in N \quad (7.3.8)\\
& \sum_{\substack{i\in N\setminus S\\ j\in S}} (\bar{z}_{ij}^k + \bar{z}_{ji}^k) \geqslant \bar{w}_u^k && \substack{S\subset N\\ |S|>0\\ u\in S} \quad (7.3.9)\\
& \sum_{i\in N^-} \bar{e}_i^k \leqslant 1 && (7.3.10)\\
& \sum_{i\in N^+} \bar{s}_i^k \leqslant 1 && (7.3.11)\\
& \bar{z}_{ij}^k \in \mathbb{B} && i,j\in N \quad (7.3.12)\\
& \bar{w}_i^k \in \mathbb{B} && i\in N \quad (7.3.13)\\
& \bar{s}_i^k \in \mathbb{B} && i\in N^+ \quad (7.3.14)\\
& \bar{e}_i^k \in \mathbb{B} && i\in N^- \quad (7.3.15)
\end{aligned}$$

Except for the depots, in a feasible solution of the pricing problem we may have three kinds of visited nodes: *integer nodes*, that are those nodes with their demands fully collected, *bridge nodes*, that are those with only one unit of demand collected, and *fractional nodes*, that are those whose demands are fractionally collected.

It may happen that nodes are visited just to collect their π_i prize, and therefore we may have more than one node in a feasible solution whose demand is not fully collected. However, we can observe that if the sequence of nodes were given, our pricing problem would reduce

to a 0-1 Knapsack Problem (KP). Therefore, as for the pricing problem presented in Subsection 3.3.2, we can exploit properties on the KP to prove that:

Theorem 7.3.1. *There always exists an optimal solution of (7.3.1) – (7.3.15) in which there is at most one fractional pickup node (resp. delivery node).*

Proof. Assume by contradiction that there exists an optimal solution in which, among the visited nodes, all demands are fully loaded on the vehicle but those of nodes i and j, which have both fractional loadings $\bar{q}_i^k$ and $\bar{q}_j^k$. Let $r = \bar{q}_i^k + \bar{q}_j^k$ be the space in the vehicle occupied by node i and node j loadings. Let us assume w.l.o.g. that node i is more efficient than node j, that is $\beta_i^{mg} > \beta_j^{mg}$. We can improve the contribution to the objective value (7.3.1) by decreasing the space occupied by the less efficient node, and increasing the space of the most efficient. The argument iteratively extends to solutions in which the number of nodes is higher than two. □

Observation 7.3.2. *There always exists an optimal MP solution, in which each selected column contains at most one fractional node with more than one unit of demand collected.*

In fact, each MP column can be found by solving the pricing problem, and thus by adding columns with at most one node with fragmented quantity.

Furthermore:

Observation 7.3.3. *In any optimal solution of the pricing problem, the prize β_j of a fractional node j is less than or equal to the prize β_i of any integer node i, and it is greater than or equal to the prize β_u of any bridge node u, that is*

$$\beta_i \geqslant \beta_j \geqslant \beta_u.$$

The proof directly follows the proof of Theorem 7.3.1.

PRICING ALGORITHM. The pricing problem is a variant of the *Resource Constrained Elementary Shortest Path Problem (RCESPP)*, that it is known to be NP-Hard. However, a recent survey [62] reviews very effective methods that solve the RCESPP by means of labelling algorithms. In [8], the authors use a discretization approach to solve the pricing problem for the SDVRP, solving an RCESPP on an extended graph with d_i nodes for each station i. The drawback of such approach is that the size of the graph grows with both the number of

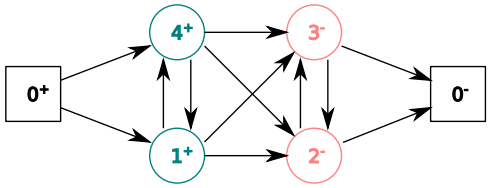

Figure 13: Example of auxiliary graph used to solve the pricing problem: pickup nodes are unreachable from the time the vehicle visits the first delivery node.

nodes and the magnitude of demand coefficients. Another approach to solve the pricing problem consists of a nested column generation [79], in which also the pricing problem is solved by branch-and-price. In [47] the authors exploit such approach to solve the SPSDVRP on a maritime crud oil transportation problem in which the first level pricing problem is considered too complex to be solved efficiently by a dynamic programming algorithm. Instead, in [35] the author solves the pricing problem of a SDVRP with Time Windows by means of a labelling algorithm. In such a problem, the routes generated have at most one split demand, and the author devises an algorithm that generates three different labels at each visit, depending on the residual capacity of the vehicle: one label when the node is visited but no unit of demand is collected, one with a fractional loading, and one in which the demand is fully collected. In fact, except for the handling of time windows, the pricing problem of [35] is very similar to ours.

Therefore, inspired by [35], and adapting techniques from [65], we propose a dynamic programming algorithm devised for our specific case.

Let us consider a single vehicle with a limited capacity C and a particular graph $\tilde{G} = (\tilde{N}, \tilde{A})$, where $\tilde{N} = N \cup \{0^+, 0^-\}$. All nodes $i \in \tilde{N}$ have a demand d_i but nodes 0^+ and 0^-. When a node i is visited, its prize π_i is collected, while a prize β_i^{mg} is collected for each unit of demand of node i loaded on the vehicle.

We denote as $\tilde{A}$ the set of arcs of the graph, that is composed by

- arcs from 0^+ to all nodes $i \in \tilde{N}^+$;
- arcs from nodes $i \in \tilde{N}^+$ to all nodes $j \in \tilde{N}$;
- arcs from nodes $i \in \tilde{N}^-$ to all nodes $j \in \tilde{N}^- \cup \{0^-\}$.

An example of the graph is depicted in Figure 13.

For each arc (i, j) we have a cost

$$c_{ij} = \begin{cases} \gamma_j^{mg}, \text{ if } i = 0^+ \\ \gamma_i^{mg}, \text{ if } j = 0^- \\ \alpha_{ij}^{mg}, \text{ otherwise} \end{cases}$$

Furthermore, capacity is always replenished when travelling from a pickup node to a delivery node.

The objective of the problem is to find a minimum cost path that goes from 0^+ to 0^-.

The pricing problem is a variant of RCESPP in which we have a resource r indicating the residual space in the partial path, and a set V indicating which stations have already been visited. We stress that information on the amount of bikes loaded or unloaded on the truck at each node are not needed, as we never pass through the same node twice. Let us remark also that, because of the particular structure of the graph, each path has two distinguishing features: first, it has at least one pickup and one delivery node, and second, no pickup node is visited after a delivery node.

Therefore our label is a tuple (i, c, r, V, f), where i is the ending node of a partial path, c is the cost of such partial path, and f indicates that a fractional node has been visited.

INITIALIZATION. We initialize our algorithm with a label λ representing a partial path starting from 0^+ with no initial cost, full resources, and no fractional node, that is

$$\lambda = (0^+, 0, C, \{0^+\}, 0).$$

LABEL EXTENSION. As a selection strategy, at each iteration we select the most profitable label $\lambda' = (i, c', r', V', f')$ with minimum partial cost. We extend such label to all neighbours j of i such that $j \notin V'$, in three different ways:

INTEGER: j is selected as an integer node, fully collecting its demand. The resulting label $\lambda'' = (j, c'', r'', V'', f')$ has a partial cost $c'' = c' + \tilde{c}_{ij} - \beta_j^{mg} \cdot d_j - \pi_j$, residual capacity $r'' = r' - d_j$, and $V'' = V' \cup \{j\}$;

BRIDGE: j is selected as a bridge node, collecting only one unit of its demand. The resulting label $\lambda'' = (j, c'', r'', V'', f')$ has a partial

cost $c'' = c' + \tilde{c}_{ij} - \beta_j^{mg} - \pi_j$, residual capacity $r'' = r' - 1$, and $V'' = V' \cup \{j\}$;

FRACTIONAL: only if no fractional node is selected yet, j is selected as a fractional node. It is not possible to determine a priori the collected demand of j, and therefore the resulting label $\lambda'' = (j, c'', r', V'', j)$ has a partial cost $c'' = c' + \tilde{c}_{ij} - \pi_j$, $V'' = V' \cup \{j\}$. The quantity loaded in j is determined only at the end of a sequence of pickup (resp. delivery) nodes.

We also remark that if i and j are a pickup and a delivery node, respectively, the residual capacity r is fully replenished before visiting j.

DOMINANCE RULES. The drawback of our extension method is that we cannot collect the prize of a fractional node until all pickup (delivery) nodes are visited. Therefore, when checking the dominance of a given label with a fractional node, we have to ensure that such label is dominated for every value of fractional loading.

Let suppose that we are given two labels $\lambda' = (i', c', r', V', f')$ and $\lambda'' = (i'', c'', r'', V'', f'')$, with $i' = i''$ and $r' \leqslant r''$. Indeed, we may use some of the residual space of λ'' to load some units of f'', matching the residual space of λ' and therefore reducing the cost of λ''. Let us consider the two extreme cases in which we load 1 and $max_{f'} = \min\{r', d_{f'} - 1\}$ units of f' demands. In the first case, we have to load $min_{f''} = \min\{r'' - r' + 1, d_{f''} - 1\}$ units of f'' demand to match the residual space in λ', while in the second case $max_{f''} = \min\{r'' - r' + max_{f'}, d_{f''} - 1\}$. Then, we say that label λ' is dominated if it exists a label λ'' such that

$$\begin{cases} i' = i'' \\ r' \leqslant r'' \\ V'' \subseteq V' \\ c' + \beta_{f'} \geqslant c'' + \beta_{f''} \cdot min_{f''} \\ c' + \beta_{f'} \cdot max_{f'} \geqslant c'' + \beta_{f''} \cdot max_{f''} \end{cases}$$

and at least one of the inequalities is strict. Dominance rules ensure that a label is dominated only if it exists another label of a partial path ending in the same node, that has more residual capacity, has visited less nodes, and costs less for each quantity of f' loaded.

REDUCTION AND LOOKAHEAD DOMINANCE. During the process we check several condition to reduce the number of extensions. In fact, due to Observation 7.3.3 we know that an optimal solution always exists, in which no fractional node has higher prize than an integer node, nor smaller prize than a bridge node. Therefore, let us suppose that λ is the label to extend, β_j is the prize of the destination node j, and $\beta_{integer}$, $\beta_{fractional}$, and β_{bridge} are respectively the minimum prize among all integer nodes in λ, the prize of the fractional node in λ, and the maximum prize among all bridge nodes in λ.

- integer extension is forbidden when $\beta_j < \beta_{fractional}$ or $\beta_j < \beta_{bridge}$, and in general when $\beta_j \leqslant 0$;
- bridge extension is forbidden when $\beta_j > \beta_{fractional}$ or $\beta_j > \beta_{integer}$;
- fractional extension is forbidden when $\beta_j > \beta_{integer}$ or $\beta_j < \beta_{bridge}$, and in general when $\beta_j \leqslant 0$.

Furthermore, let suppose that we are given two labels λ' and λ'', both identical but for two nodes i and j. Let us suppose that i is a bridge node in λ' and a fractional node λ'', while vice versa, j is a fractional node in λ' and a bridge node in λ''. Let us suppose w.l.o.g. that $\beta_i \geqslant \beta_j$, then λ'' dominates λ'. Therefore, we aggregate fractional and bridge extension when we visit a node j with a positive prize β_j: given a label with fractional node f, if $\beta_j \geqslant \beta_f$ then a unit of f is collected, cost and residual capacity of the label are updated, j becomes the new fractional node of the label, and f becomes a bridge node. Otherwise we perform a bridge extension.

PRICER EXECUTION AND INSERTION POLICY. In a strict column generation approach, this procedure should be computed for each pair of group $g \in G$ and vehicle $m \in M$, in order to find the column with minimum reduced cost. Instead, we found profitable from a computational point of view, to perform partial pricing, and stop the pricing procedure after we find the first pair g and m yielding a negative reduced cost column. We then insert such a column in all Γ_{mg} sets.

BIDIRECTIONAL ALGORITHM. In order to further improve the performance of our pricing algorithm, we exploit bidirectional search. Since each solution is composed of a sequence of pickup nodes, and

a sequence of delivery nodes that can be treated independently, we run the algorithm on two different graphs, one composed by pickup nodes only, and one by delivery nodes only. At the end, we obtain full solutions by joining the best labels of each pair of pickup and delivery nodes.

HEURISTIC PRICING. To speed up the column generation process, we also implemented two heuristic variants of the pricing algorithm.

First we run the exact algorithm on a reduced graph, obtained by removing for each node, the set of k arcs with highest travelling cost $\tilde{c}_{ij}$, with k fixed a-priori.

Second, if the first heuristic does not find any column with negative reduced cost, we run the pricing algorithm considering columns with integer loading only. In such a way, at each extension we generate only one label in the destination node, reducing the overall number of labels.

If none of the two heuristic algorithms generate a column with negative reduced cost, we run the exact pricing algorithm.

7.3.3 *Branching rules*

When the optimal MP solution is fractional, and upper and lower bounds do not match, we check which integrality constraints in the original formulation are not satisfied and enforce them by exploring a search tree through branching rules. In our case, branching is particularly involved, as the MP is prone to symmetries.

We devised the following binary branching rule in which nodes are progressively fixed in vehicle groups. Let $\tilde{y}^k$ be the value of a variable y^k in the fractional solution of the MP, and let

$$\tilde{w}_i^{mg} = \sum_{k \in \Gamma_{mg}} \bar{w}_i^k \cdot \tilde{y}^k$$

be the fractional assignment of each node $i \in N_0$ to a group $g \in G$ of a vehicle $m \in M$. We search for a tuple $(\hat{i}, \hat{g}, \hat{m})$ that corresponds to the maximum fractional assignment in the current fractional solution, that is

$$(\hat{i}, \hat{g}, \hat{m}) \in \underset{\substack{i \in N_0 \\ g \in G \\ m \in M}}{\operatorname{argmin}} \left\{ \left| \tilde{w}_i^{mg} - \frac{1}{2} \right| \right\}.$$

If $\tilde{w}_{\hat{i}}^{\hat{g}\hat{m}}$ is fractional, then we perform binary branching: in one branch we enforce $\hat{i}$ to be always visited by vehicle $\hat{m}$ in group $\hat{g}$. In the other branch, we preclude the visit of $\hat{i}$ by vehicle $\hat{m}$ in group $\hat{g}$. Let us recall that forcing $\hat{i}$ to be visited by vehicle $\hat{m}$ in group $\hat{g}$ does not preclude the visit of $\hat{i}$ by another vehicle or in different groups.

If no fractional $\tilde{w}_{\hat{i}}^{\hat{g}\hat{m}}$ is found, we can stop branching, as an integer SDSPVRP solution can be directly found. In fact, the following holds.

Observation 7.3.4. *When no assignment to groups is fractional, there is at least one route for each vehicle that is integer.*

Proof. Let us suppose that we are given an optimal solution of the MP in a certain branching node. If no fractional $\tilde{w}_{\hat{i}}^{\hat{g}\hat{m}}$ is found, it means that for each group of each vehicle, the (potentially fractional) selected columns describe groups that are permutations of the same set of stations. Therefore, among all the permutations, only the best permutation can be selected, finding an equivalent, but integer, optimal MP solution. □

This means that for each vehicle we can arbitrary select one route among all the fractionally selected ones for that vehicle, in order to obtain a full SDSPVRP solution.

A solution obtained in such a way could still be non-integer due to loading quantities in each node, but exploiting Observation 7.2.6 we can then optimally assign such values in polynomial time, obtaining a full feasible integer solution.

7.3.4 *Branching implementation.*

First, let us remark that depot 0, is not involved in the pricing problem, and therefore if $\hat{i} = 0$ we branch by adding in the MP a constraint

$$\sum_{k \in \Gamma_{\hat{g}\hat{m}}} \bar{w}_0^k \leqslant 0 \tag{7.3.16}$$

in one branch, excluding the depot from a group, and

$$\sum_{k \in \Gamma_{\hat{g}\hat{m}}} \bar{w}_0^k \geqslant 1 \tag{7.3.17}$$

in the other branch, fixing it into the group.

Instead, if $\hat{i}$ is not 0, but a pickup or delivery node, we first exclude in one branch all columns that contains node $\hat{i}$ in group $\hat{g}$ of vehicle

$\hat{m}$, and we modify the pricing graph by removing incoming arcs in $\hat{i}$. This ensures that no further column is generated including such a node, and also improves the performances of the pricing algorithm.

In the other branch, we fix $\hat{i}$ into group $\hat{g}$ of vehicle $\hat{m}$ and, to ensure that the MP selects columns with such a feature, we add the constraint

$$\sum_{k \in \Gamma_{\hat{g}\hat{m}}} \bar{w}_{\hat{i}}^{k} \geqslant 1. \tag{7.3.18}$$

This constraint, in turn, introduces a new dual variable that must be considered in the pricing problem. Therefore, when we visit node $\hat{i}$ during dynamic programming extension, we collect a prize corresponding to that dual variable. Let us remark that such a detail does not change the structure of the pricer.

7.3.5 *Additional inequalities*

According to literature [22]:

Theorem 7.3.2. *For any optimal solution, given two pickup (or equiv. two delivery) nodes* i *and* j*, the number of times arc* (i, j) *is used plus the number of times arc* (j, i) *is used is less than or equal to* 1*.*

We extend this theorem and prove the following:

Theorem 7.3.3. *For any optimal solution, given two pickup (or equiv. two delivery) nodes* i *and* j*, the number of times* i *and* j *are in the same group is less than or equal to* 1*.*

Proof. Let us consider w.l.o.g. an optimal solution in which two vehicles visit nodes i and j in the same group. Let a_i and a_j be the number of bikes loaded (unloaded) by the first vehicle on nodes i and j, and let b_i and b_j be the number of bikes loaded (unloaded) by the second vehicle. It may happen that:

- $a_i \geqslant b_j$: in this case we can set new loading values $a'_i = a_i - b_j$, $a'_j = a_j + b_j$, $b'_j = 0$ and $b'_i = b_i + b_j$. The quantity previously loaded from b_j is then loaded from a'_j, while the overall load of each vehicle does not change since a_i is decreased by b_j and b_i is increased by b_j;

- $a_i < b_j$: in this case $a'_i = 0$, $a'_j = a_j + a_i$, $b'_j = b_j - a_i$ and $b'_i = b_i + a_i$. The quantity previously loaded from a_i is then

loaded from b'_i, while the overall load of each vehicle does not change since b_i is decreased by a_i and a_j is increased by a_i.

In both cases one vehicle visits a node without any operation and therefore the corresponding arc is removed.

□

We remark that the theorem holds also when nodes are visited by the same vehicle.

Corollary 7.3.4. *Let v_{ij}^{mg} be the binary variable that is 1 if pickup (delivery) node i is visited together with pickup (delivery) node j in group g of vehicle m; then constraints*

$$\sum_{\substack{m\in M\\ g\in G}} v_{ij}^{mg} \leqslant 1 \tag{7.3.19}$$

are valid inequalities for model (7.2.1) – (7.2.30).

In the MP, Constraints (7.3.19) become

$$\sum_{\substack{m\in M\\ g\in G\\ k\in \Gamma_{mg}}} \bar{v}_{ij}^{k} \cdot y^{k} \leqslant 1 \tag{7.3.20}$$

and their corresponding dual variables must be taken into account into the pricing problem. In fact, let ξ_{ij} be the dual variables of Constraints (7.3.20), we first add the terms

$$\sum_{\substack{i,j\in N^{+}\\ i<j}} \xi_{ij} \cdot \bar{v}_{ij}^{k} + \sum_{\substack{i,j\in N^{-}\\ i<j}} \xi_{ij} \cdot \bar{v}_{ij}^{k}$$

to the objective function (7.3.1). Then, we also add constraint

$$w_i + w_j \leqslant v_{ij} + 1$$

to the pricing problem for each pair of pickup (delivery) nodes such that $i < j$.

For what concerns the implementation of the pricing algorithm, we modify the extension in such a way that when we extend to a node j, we add to the new label the sum of the costs associated to the node j, that is

$$\sum_{\substack{i\in N^{+}\\ i<j}} \xi_{ij}$$

if j is a pickup node, and

$$\sum_{\substack{i \in N^- \\ i<j}} \xi_{ij}$$

if j is a delivery node.

Let us remark that such inequalities do not change the structure of the pricing algorithm. In fact, a label can be dominated only by a label that visited a subset of its nodes. Therefore, all the prices collected by the latter, are also collected by the former.

7.3.6 *Infeasibility detection*

During the exploration of the branching tree, we may run into nodes corresponding to infeasible partial solutions. Indeed, solving the MP relaxation of such a node would reveal the infeasibility, but this may require several iterations of column generation. Therefore we perform a first feasibility check on the node, in order to detect its infeasibility before computing its continuous relaxation. If the test succeeds, then the current node may lead to a feasible solution, and therefore the relaxation is computed. Otherwise, the node is simply discarded.

In details, let us suppose to have a partial solution obtained through branching, and let suppose that for each group $g \in G$ and vehicle $m \in M$, we have a set F^{mg} of nodes that are fixed into such a group, and a set E^{mg} of nodes that are excluded. All nodes that do not belong to any of the two sets are free nodes that may or may not be visited in such a group.

We then build a graph similar to the one in Subsection 7.2.2, in which we have a source node s, a depot node t, a node f_i^+ for each pickup node i, a node f_i^- for each delivery node i, and two nodes p^{mg} and d^{mg} for each available group of each vehicle.

Now, let us define $\tilde{d}_i$ as the maximum quantity of demand of node i that is not fixed into any group, that is if a node is fixed in one group $\tilde{d}_i = d_i - 1$, and in general

$$\tilde{d}_i = d_i - \sum_{\substack{g \in G \\ m \in M}} |F^{mg} \cap \{i\}|$$

We add arcs from s to f_i^+ and from f_i^- to t with capacity $\tilde{d}_i$. These arcs limit the quantity of demand we are free to distribute for nodes that have been fixed in groups.

Then, for each pickup node i, group g and vehicle m add an arc from f_i^+ to p^{mg} if $i \notin E^{mg}$, and respectively for each delivery node i, group g and vehicle m add an arc from d^{mg} to f_i^- if $i \notin E^{mg}$. Also, for each pickup node $i \in F^{mg}$, add an arc with capacity 1 from s to node p^{mg}, and respectively for each delivery node $i \in F^{mg}$, add an arc with capacity 1 from d^{mg} to node t. These arcs impose that to cover the demand of a station, some flow must pass through the groups in which such station has been fixed.

Finally, from each node p^{mg} add an arc to d^{mg} with capacity C, and from each node d^{mg} add an arc to p^{mg+1}, with infinite capacity.

An example of the graph is show in Figure 14, representing the partial solution in which we have at most 2 groups for each vehicle, and

- node 1 is fixed in group 1 of vehicle 1;
- node 5 is fixed in group 2 of vehicle 2;
- node 2 is fixed in group 2 of vehicle 1;
- node 10 is fixed in group 2 of vehicle 2.

Furthermore, node 1 and 4 are excluded respectively from groups 1 and 2 of vehicle 2, while 7 and 9 can be visited only by vehicle 2. Node 2 can be visited only by vehicle 1, while node 3 and 10 care excluded respectively from group 2 of vehicle 2 and group 2 of vehicle 1.

Observation 7.3.5. *If the maximum flow going from s to t is less than the overall pickup (delivery) demand, the partial solution of the branching node is infeasible.*

In fact, if the maximum flow is less than the overall demand, it means that the demand of at least one node has not been satisfied.

The contrary does not necessarily hold: in fact, if a partial solution is infeasible, it may still pass the feasibility check, because it is not possible to ensure that for each vehicle it exists a route that does not exceed resource T.

7.4 EXPERIMENTAL ANALYSIS

We implemented our algorithms in C++, using the SCIP framework [1] version 3.0.2. The LP subproblems were solved using the simplex algorithm implemented in CPLEX 12.4 [36]: the framework automatically switches between primal and dual methods. To obtain good upper bounds we also included a generic rounding heuristic from SCIP.

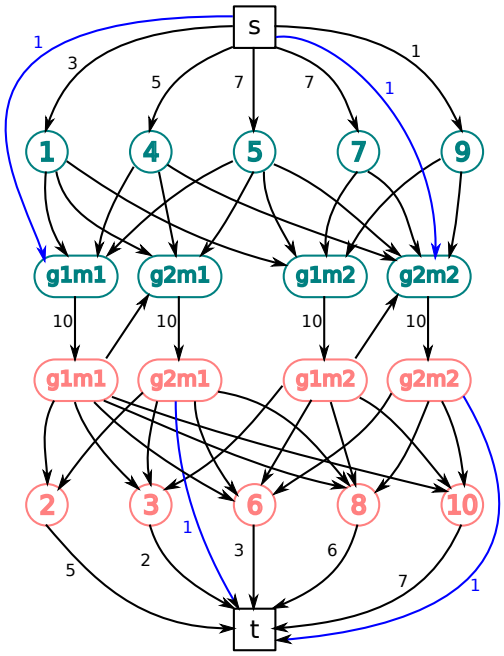

Figure 14: Example of graph for feasibility check (infinite capacities are omitted).

Unfortunately a real instance would consists of hundreds of stations, and would be out of reach for our methodology. Therefore, as a benchmark we considered the set of instances used in [22] for the SPSDVRP with 10 nodes. Each instance describes a randomly generated network with nodes located in the two-dimensional space $[-500, 500] \times [-500, 500]$, with the depot located at $(0, 0)$. Travelling costs c_{ij} are computed as the Euclidean distance between nodes i and j. Each station has a demand randomly generated between $[-10, 10]$, where positive values define pickup nodes, and negative values define delivery nodes, and with the sum of the pickup demands is equal to the sum of delivery demands. The vehicle capacity C is set to 10, and the number of available vehicles is 5.

The time limit resource was undefined in the original instances. To obtain a fair comparison with previous approaches from the literature we set $T = 10$, and each $t_{ij} = 1$. This means that each vehicle can visit at most 9 stations before going back to the depot.

We compared our results with the exact algorithm described in [22], and sketched in the Introduction, that is able to find optimal solutions for all instances with 10 nodes. We compiled the original C++ source code with additional optimization flags, linking it to CPLEX version

12.4 libraries instead of version 12.0. A few additional coding tweaks were needed to ensure correct runs. Both algorithms have then been executed on a machine with Intel(R) Core i7-2640M at 2.80 GHz and 8 GB of memory in single thread mode.

In the remainder we refer to our exact branch-and-price algorithm as BPA, while we refer to the algorithm in [22] as MH.

7.4.1 *Column generation profiling*

In a first round of experiments we performed a profiling of our algorithm in order to detail how the time is spent during the computation of a lower bound.

In Table 12 we report the results obtained at root node. For each instance we report the total time spent solving the LP during column generation, and for each type of pricer we report the number of calls, the number of generated variables, and the total running time. Results show that most of the time is spent in solving the LPs. Heuristic pricers are effective, reducing the number of columns that must be generated by the exact pricer.

In Table 13 we report the results obtained while solving the problem to proven optimality. Still, solving the LP is usually the most time consuming operation. In the overall process, the heuristic pricing it is not effective, while the heuristic integer pricing still manages to reduce the number of calls to the exact pricer.

7.4.2 *Root lower bound*

In the second round of experiments we compared both the quality of the lower bound and the efforts required to obtain it.

In Table 14 we report for each instance the gap and the time needed to MH to compute a lower bound, the lower bound given by BPA at root node, and the lower bound and time needed to BPA to obtain the same bound of MH. Technically speaking, after setting such a target, it often happens that BPA even improves the bound of MH. The results show that BPA computation times at the root node are orders of magnitude smaller than the MH ones. However, the average lower bound given by the BPA at the root node is worse than that of MH. However, during the exploration of the branching tree, our algorithm provides on average a better lower bound than MH four times faster.

Instance		RMP	Heuristic pricer			Integer pricer			Exact pricer		
graph	$\overline{d}$	t (s)	calls	vars	t (s)	calls	vars	t (s)	calls	vars	t (s)
n10q10a	52	0.44	23	1028	0.03	18	1075	0.06	8	275	0.13
n10q10A	48	0.22	15	778	0.02	10	475	0.01	5	275	0.1
n10q10b	60	0.22	18	871	0.03	12	525	0.06	3	50	0.11
n10q10B	32	0.27	18	903	0.01	12	250	0	5	325	0.07
n10q10c	64	0.31	20	701	0.05	17	1475	0.06	8	975	0.27
n10q10C	44	0.14	8	778	0.02	6	250	0	5	525	0.08
n10q10d	54	0.36	22	725	0.01	19	1400	0.03	6	500	0.1
n10q10D	42	0.21	12	1228	0.02	10	525	0.02	4	75	0.21
n10q10e	58	0.26	14	928	0.01	10	575	0	5	625	0.07
n10q10E	58	0.35	17	728	0.02	14	450	0.02	6	550	0.1
n10q10f	50	0.19	12	1323	0.03	9	1400	0.1	3	50	0.19
n10q10F	36	0.28	16	1153	0.01	13	825	0.06	6	200	0.33
n10q10g	52	0.28	24	897	0.06	19	1075	0.05	3	50	0.12
n10q10G	34	0.19	12	1203	0.02	8	475	0.01	4	275	0.15
n10q10h	48	0.18	13	853	0	10	125	0.01	7	250	0.09
n10q10H	62	0.28	17	1303	0.03	14	1300	0.04	5	100	0.13
n10q10i	44	0.24	18	1253	0.03	15	425	0.02	4	125	0.37
n10q10I	60	0.19	13	776	0.02	10	425	0.01	2	25	0.04
n10q10j	56	0.41	29	900	0.04	23	950	0.04	7	525	0.07
n10q10J	42	0.24	17	1128	0.02	12	350	0.01	7	425	0.08

Table 12: Time spent during the computation of a lower bound at root node

Instance		MP	Heuristic pricer			Integer pricer			Exact pricer		
graph	$\overline{d}$	t (s)	calls	vars	t (s)	calls	vars	t (s)	calls	vars	t (s)
n10q10a	52	12.01	454	1628	0.22	436	12775	1.61	296	5550	4.2
n10q10A	48	1.6	91	878	0.01	85	1450	0.13	61	2875	0.72
n10q10b	60	6.35	286	1246	0.14	269	9975	1.2	140	1375	1.93
n10q10B	32	7.42	284	1503	0.13	258	3775	0.48	206	9775	2.7
n10q10c	64	0.87	49	826	0.03	43	1950	0.11	27	1600	0.58
n10q10C	44	50.98	1226	2278	0.71	1184	12200	3.19	994	20350	13.28
n10q10d	54	572.42	1988	1850	0.92	1960	50125	23.46	1337	41275	38.21
n10q10D	42	77.59	2469	2003	1.5	2441	14850	11.76	2149	10800	78.47
n10q10e	58	72.69	1699	1853	0.87	1669	10500	3.62	1473	25625	17.99
n10q10E	58	37.56	1063	1853	0.52	1030	8850	2.13	876	21200	10.25
n10q10f	50	49.27	1108	1973	0.64	1083	29625	14.22	683	8500	30.99
n10q10F	36	3.89	134	1428	0.06	127	3650	0.2	87	2600	2.38
n10q10g	52	5.3	304	1447	0.21	280	8450	1.12	151	1500	2.72
n10q10G	34	2.51	135	1428	0.07	124	2150	0.37	91	3700	3.49
n10q10h	48	7.51	410	1753	0.26	377	4825	0.78	306	4075	4.95
n10q10H	62	10.96	607	1778	0.33	591	9075	2.1	441	6725	12.91
n10q10i	44	14.04	678	1928	0.42	657	7675	2.1	533	11200	23.67
n10q10I	60	10.37	506	1626	0.3	474	10250	1.51	316	4775	4.15
n10q10j	56	44.43	1253	1775	0.49	1223	22425	5.62	891	14550	11.94
n10q10J	42	7.24	417	1603	0.2	401	3150	0.39	350	6075	3.24

Table 13: Time spent during column generation procedures

Instance		**MH**		**BPA**			
				root node		gap limited	
graph	$\overline{d}$	Gap (%)	t (s)	Gap (%)	t (s)	Gap (%)	t (s)
n10q10a	52	2.98	33.8	7.22	0.66	2.77	10.15
n10q10A	48	0.00	32.04	9.09	0.36	0.00	2.54
n10q10b	60	2.49	39.31	12.79	0.43	1.92	9.07
n10q10B	32	1.98	56.53	5.75	0.38	1.85	9.92
n10q10c	64	0.00	30.35	0.10	0.70	0.00	1.59
n10q10C	44	4.11	297.73	16.48	0.25	4.08	24.27
n10q10d	54	6.90	31.82	15.61	0.54	6.81	26.74
n10q10D	42	1.64	63.92	31.24	0.47	1.64	160.02
n10q10e	58	4.01	103.65	10.50	0.36	3.98	44.13
n10q10E	58	0.39	73.12	14.73	0.52	0.37	49.88
n10q10f	50	5.62	24.71	20.01	0.53	5.22	35.75
n10q10F	36	0.03	42.65	4.76	0.68	0.00	6.93
n10q10g	52	2.55	29.39	18.38	0.55	2.07	10.08
n10q10G	34	6.35	65.21	11.57	0.37	3.53	4.00
n10q10h	48	3.07	109.46	13.71	0.29	2.96	8.10
n10q10H	62	4.96	92.53	10.54	0.50	4.72	10.31
n10q10i	44	0.47	442.69	31.69	0.71	0.45	42.67
n10q10I	60	6.78	38.54	18.48	0.27	6.42	6.55
n10q10j	56	1.79	38.58	15.11	0.61	1.78	55.00
n10q10J	42	0.00	392.1	37.71	0.36	0.00	12.45
Average		2.81	101.91	15.27	0.48	2.53	26.51

Table 14: Lower bounds on instances with 10 stations and $d_i \leqslant 10$

7.4.3 *Upper bound*

Third, we compared the quality and the time needed to compute a good upper bound for the problem. Let us remark that our algorithm is not intended to be used as a heuristic, and in fact we did not implement a specific heuristic for such a problem, but we rather used an off-the-shelf generic rounding heuristic available in the SCIP framework. Such a heuristic iteratively rounds fractional variables trying to recover from infeasibility whenever a constraint is violated.

For what concern MH, the algorithm runs its meta-heuristic before and after computing a lower bound to the problem. In our tests we considered the solution given by the algorithm after finishing the first meta-heuristic round. Instead, for what concern BPA we perform two different analyses: first we stopped the algorithm when a first solution is found. Second, we ran the algorithm until the gap between the upper bound and the known optimal solution is zero.

In Table 15 we report for each instance class and for each method, the gap between the corresponding upper bound and the optimal solution, and the time required for the computation. For what concern the BPA we report the number of nodes analysed before reaching its best UB, and the number of nodes and computation time needed to find the optimal solution. The results show that a first solution is found quickly, is usually good, and sometimes even optimal. Furthermore, the optimal solution is usually found in the early phases of the branching tree exploration, when only few nodes have been analysed. Indeed, BPA is able to retrieve an optimal solution within the same amount of time required by MH when used as a heuristic.

7.4.4 *Solving instances to proven optimality*

Finally, we compared MH and BPA in solving instances of SPSDVRP to proven optimality. In table 16 we report our results. For each instance we report the sum of the demands $\overline{d}$, the computation time for both algorithms and the number of nodes explored by our BPA.

As we can see, BPA is on average much faster than MH. In fact it is slower only for three instances, and requires more than 5 minutes of computation only for one instance.

Instance		BPA					MH	
		optimal sol.		first sol.				
graph	$\overline{d}$	nodes	t (s)	Gap (%)	nodes	t (s)	Gap (%)	t (s)
n10q10a	52	91	9.33	5.86	75	8.03	0.00	23.10
n10q10A	48	15	1.92	0.00	15	1.92	0.00	25.32
n10q10b	60	110	10.81	23.37	13	3.20	0.00	30.36
n10q10B	32	115	12.10	38.69	4	1.63	1.11	24.55
n10q10c	64	3	0.93	0.00	3	0.93	0.00	26.20
n10q10C	44	86	7.87	7.31	66	6.16	0.00	29.03
n10q10d	54	67	12.65	5.18	34	4.73	0.00	23.58
n10q10D	42	1465	144.00	1.88	107	13.37	2.31	23.26
n10q10e	58	531	51.81	12.90	101	14.07	0.60	24.37
n10q10E	58	227	28.85	22.25	40	6.86	0.92	24.95
n10q10f	50	307	52.64	10.61	34	7.71	0.00	19.37
n10q10F	36	20	3.92	0.00	20	3.92	1.05	23.33
n10q10g	52	115	10.16	21.03	63	6.74	0.00	23.45
n10q10G	34	40	6.41	8.41	12	2.55	4.48	23.10
n10q10h	48	127	9.90	1.42	27	2.36	0.00	24.72
n10q10H	62	272	25.45	3.54	60	7.75	0.00	28.27
n10q10i	44	374	42.57	14.93	13	3.96	0.00	27.37
n10q10I	60	217	18.15	9.46	51	5.80	1.59	24.69
n10q10j	56	426	51.65	24.76	90	14.77	0.00	28.17
n10q10J	42	42	2.70	0.00	42	2.70	0.00	20.04
Average			25.19	10.58		5.96	0.60	24.86

Table 15: Upper bounds on instances with 10 stations and $d_i \leqslant 10$

Instance		MH	BPA	
graph	$\overline{d}$	t (s)	nodes	t (s)
n10q10a	52	112.53	189	**20.21**
n10q10A	48	43.87	24	**2.69**
n10q10b	60	63.58	110	**10.82**
n10q10B	32	115.24	115	**12.11**
n10q10c	64	45.67	7	**1.63**
n10q10C	44	1214.69	652	**79.39**
n10q10d	54	**141.70**	724	711.49
n10q10D	42	583.03	1886	**187.05**
n10q10e	58	343.51	989	**113.29**
n10q10E	58	211.41	537	**59.69**
n10q10f	50	**93.56**	551	106.36
n10q10F	36	60.26	43	**6.79**
n10q10g	52	99.01	119	**10.45**
n10q10G	34	292.13	44	**6.70**
n10q10h	48	2732.18	201	**14.77**
n10q10H	62	909.23	313	**29.09**
n10q10i	44	3596.12	388	**44.04**
n10q10I	60	144.44	217	**18.17**
n10q10j	56	**57.98**	616	75.25
n10q10J	42	393.80	235	**12.22**
Average		562.70		76.11

Table 16: Results on the instances with 10 stations and $d_i \leqslant 10$

7.5 CONCLUSIONS

In this chapter we proposed an exact method to tackle a SPSDVRP arising on a bike-sharing system. That corresponds to the problem of balancing bikes on a network using a fleet of homogeneous vehicles that may split the demands of each customer on the network.

In order to reduce its complexity, we modelled the problem by decomposing routes into substructures called *groups*. Such groups help to discard sub-optimal configurations and to limit problem symmetries. They favour also a decomposition approach from an algorithmic point of view.

To improve the lower bound given by our model, we produced an extended formulation by using Dantzig-Wolfe decomposition. We solved the linear relaxation of the extended formulation using column generation techniques, and integrated such a procedure into a branch-and-price framework. Our ad-hoc pricing algorithms, branching rules, feasibility detection routines and additional cuts proved to be computationally useful. As an overall assessment, our approach turns out to be faster and more flexible than competitors.

8

CONCLUSIONS

In this thesis we proposed new approaches to tackle optimization problems with fractional resources. We investigated different approaches to efficiently solve variants of both packing and vehicle routing problems in which splits are allowed. We designed exact and heuristic methods and performed experimental studies in order to evaluate the performances of the algorithms.

For what concerns the Bin Packing Problem with Item Fragmentation, we proved that optimal solutions with primitive structure always exist. We performed a theoretical investigation, exploited properties on the structure of the solution, and applied Dantzig-Wolfe decomposition method, to obtain both an exact algorithm and fast heuristics for such problems. We also identified classes of instances that are computationally harder to solve.

We then further extended theoretical properties to devise new models that embed the split component. By using Dantzig-Wolfe decomposition method and column generation techniques, we obtained algorithms of orders of magnitude faster. Also, we proved that the underlying theory can be extended to different variants of the problem, obtaining a resolution framework for this category of problems.

We also addressed a new variant of the BPPIF, in which a fixed cost is paid whenever an item is split, no matter how many times. Such a variant is not included in the ones that can be solved with our framework, but still, by exploiting properties on the structure of the optimal solution, it is possible to reduce the problem to a pure combinatorial one, improving the performances even when using a general purpose solver.

For what concerns the Split Pickup Split Delivery Vehicle Routing Problem on a bike-sharing system, we observed that the main issue using column generation techniques in such problem, is the hardness of the pricing procedure when the amount of demand to be served is computed with the vehicle route. We devised a new mathematical programming model that exploits properties on substructures of routes, called groups. Then, we applied Dantzig-Wolfe decomposition method to such a model in order to obtain an extended formula-

tion with a manageable pricing problem. Results obtained with such approach seems proved that our methodology is promising and can solve small instances faster than competitors.

We remark that not every optimization problem becomes harder when resource splitting is allowed. For instance, in capacitated facility location problems, allowing fractional assignments of clients to facilities yields more tractable formulations. This is not therefore an intrinsic property of every MILP, but rather a feature of a class of combinatorial optimization problems. Indeed, in some cases it is needed to link together continuous and integer variables via *big-M* like constraints, or to search for similar modelling workarounds, which are known to reduce the quality of continuous relaxations [27]. This is certainly the case of the packing and routing problems addressed in this thesis, but also of problems in statistics, like the minimization of the ℓ_0 and ℓ_1 norms [18, 83] in sparsity enhancement techniques, just to mention a notable example. Our methods may therefore apply successfully to many problems in this area.

We conclude that a successful approach in solving combinatorial optimization problems with fractional resources, typically consists in combining two ingredients: first, theoretical results on the structure of the problem, allowing to defer fractional decisions at the end of the optimization process; second, decomposition methods allowing to limit the complexity of the problem by splitting into smaller and easier sub-problems. In particular, our approach of postponing fractional decisions after the optimization process often yields easier pricing problems and reduces the number of branching decisions needed to reach an integer solution. Such approach is far superior to a simpler one in which a general purpose solver is used giving priority to branch on certain sets of variables.

Future research may extend our approach to other problems with fractional resources. Also, given a problem with fractional resources, it is unknown (a) if it always exists a structure of the solutions that allows a pure combinatorial version of the problem, (b) if such reformulation can be obtained with an automatic methodology, and (c) if the reformulation is always better than the original one. In fact, such methodology could be eventually included in general purpose solvers, to be activated only by need if special problem structures are detected, improving their efficiency on a class of problems that is currently too hard to be solved.

Part III

APPENDIX

A

DYNAMICALLY NEGOTIATING TIME SLOTS IN ATTENDED HOME SERVICE DELIVERY

Home service optimization is becoming a key issue in many industrial sectors [82]. For instance, it is common practice for large technology stores in Europe to offer both the delivery of products at home after purchase, and additional professional services like installation and setup, either for free or at a charge. On one hand, the customer must wait at home for the service to be provided, and is therefore interested in having very well defined, guaranteed, service time slots; he is also interested in choosing as much as possible the placement of the slots that best suits his needs. On the other hand, retailers that must provide such a service are interested in minimizing costs, that mainly consist in limiting the number of operators involved in the home services. Often, the number of operators employed is even fixed in advance by the retailer, who is then interested in offering the best possible level of service to the customer, taking into account the limited service possibility of her operators. In turn, the skill of matching negotiated service time windows is crucial for a store reputation, being one of the main items in the score, and one of the most commented topics, given by users in online recommendation systems [74].

In this context crucial decisions must be taken at different levels, like the tactical definition of timeslots and the operational scheduling of the operators; different strategies have been developed, typically trying to trade time slot flexibility with price incentives and discounts [17] or drawing similar methodologies from revenue management [3]. Any approach agree on a common principle: while a service time slot may be negotiated with some degree of flexibility, missing a fixed appointment is perceived as a strong disservice by the customer.

Later, more focused investigations on the tactical definition of time slots have appeared in the literature [2]. Recently, Desaulniers and Spliet designed a system in which the customer set is known, but their demand is not: they negotiate multiple time windows; routing is then performed only when actual demand becomes known [72]. Such time slot selection problems share some features also with Dynamic Vehicle Routing Problems, in that the set of customers appears in an online

fashion; a recent review can be found in [60]. Finally, we mention the analogies of negotiating service delivery time with that of accepting transportations or not in dynamic dial-a-ride problems [51, 16].

None of the works in the literature, however, face at the same time the problem of deciding the service time slots (a) at an operational level of detail, that is explicitly producing hard time windows, together with a suitable scheduling for an operator to meet them, (b) in an online fashion, that is answering to each customer at his arrival time, without assuming any distribution on future customers, and without the possibility of re-negotiating the slots at later time, and (c) with realtime performances, that is with computational methods yielding decision support options in small fractions of seconds.

Hence, in this thesis we formalize a time slot allocation problem and we design decision support tools, relying on graph models and combinatorial optimization algorithms, that are able to cope with issues (a), (b) and (c) simultaneously. We also introduce suitable indicators, highlighting key level of service factors in the form of quality measures. Finally, we perform a detailed experimental campaign, trying to show through computer simulations the trade-off that can be achieved between different level of service measures.

In Section A.1 we formalize the problem and introduce our mathematical notation; in Section A.1.2 we model the level of service by three quality measures. In Section A.2 we discuss our online negotiation policies. In Section A.3 we report on our experimental campaign. Finally, in Section A.4 we briefly draw some conclusions.

A.1 MODELS AND METHODS

Our time slot allocation problem involves three actors: a set of *customers*, that ask for a certain home service at a particular day and time, an *operator* that performs such a service, and a *service provider* that acts as an interface between customers and operators, by negotiating service slots and by creating daily schedules.

A.1.1 *Service scheduling model*

From the point of view of the service provider, we model the service scheduling as the following two-step process.

TIME SLOT NEGOTIATION. Let I be a sequence of customers. During the day, each customer $i \in I$ appears at the provider's counter in an online fashion, asking for the delivery of a service in a desired time window $[a_i, b_i]$ of a certain day. The provider can either directly accept the customer's request, negotiate an alternative service time window $[c_i, d_i]$ on the same day, or negate service in the desired day, and defer to alternative service days. Once an agreement is reached, no change in the negotiated day and time window is allowed: it is mandatory for the provider to meet the service slot they agreed.

ROUTING. Then, at the end of the day, a routing for a single operator is computed, in order to service the accepted customers in their negotiated time windows. The scarce resource is time: the operator has a limited working shift, that without loss of generality we indicate as $[0, T]$. Moving from the location of a customer i to that of a customer j takes D_{ij} units of time, that is D represents the distance matrix between customers; once at destination, we consider the time needed to perform service at i to be known, and we denote it as w_i. Therefore, computing a feasible routing amounts to find a feasible solution to a Traveling Salesman Problem with Time Windows (TSPTW) [71], considering the operator as a vehicle leaving a depot at time 0, visiting once each of the accepted customers within their negotiated time windows, and going back to the depot before time T.

By design, the two service scheduling steps are of radically different nature. The routing problem is an *offline* problem, that can be solved by overnight computations once per day. A provider may assume the traveling cost to be not an issue, as the operator is expected to move in a rather small urban area, or may provide an additional suitable cost function. In any case, once the set of customers and their service time windows is fixed, the problem of finding an optimal schedule turns out to be a traditional TSPTW, for which very efficient exact algorithms are presented in the literature [10]. The time slot negotiation, instead, is an *online* problem to be solved with *realtime* efficiency: the sequence of customers is not known in advance, and every time a new customer appears, the provider has to be able to give answers in fractions of seconds. Since at this stage it is crucial not to miss a fixed service, the provider needs a procedure $\sigma()$ taking in input the desired service day and time window of a new customer, and the set of accepted customers for that day and their negotiated time window,

and producing as output either an alternative time window or a 'null' value, indicating that the new customer cannot be serviced in the desired day; in the latter case we say for the sake of simplicity that the customer is *rejected*, although the actual behavior of the provider is to repeat the negotiation process on a different day. More formally, let $\bar{I}$ be the set of accepted customers for the desired service day, $[c_i, d_i]$ be the negotiated time window for each customer in $\bar{I}$, and $[a, b]$ be the desired time window of the new customer j, a procedure

$$\sigma(\bar{I}, \bigcup_{i \in \bar{I}} [c_i, d_i], D, j, [a, b]) \to [c, d]$$

is needed in the decision making process, where $[c, d]$ represents an alternative service time window for the new customer, or encodes a 'reject' value $[-\infty, +\infty]$.

A.1.2 *Quality measures*

Of course, among all possible *feasible* time slot allocations, the service provider may search for ones providing *high level of service* $\mathbb{L}$. There are many ways of defining good plans. In the following we describe three possible measures.

ACCEPTANCE RATE. First, the provider may be interested in rejecting as few customers as possible, as changing the service delivery day is usually perceived by a customer as the worst level of service. We therefore define the rate of acceptance quality measure as follows:

$$\mathbb{L}^a = \frac{|\bar{I}|}{|I|}.$$

AMOUNT OF TIME SHIFT. If alternative time windows are proposed to a customer, a certain worsening in her perceived level of service is introduced. The most intuitive way of measuring such a worsening is by means of average amount of shift of the negotiated time window with respect to the desired one, that is

$$\mathbb{L}^s = \frac{\sum_{i \in \bar{I}} |(a_i + b_i)/2 - (c_i + d_i)/2|}{|\bar{I}|}.$$

AMOUNT OF WINDOW ENLARGEMENT. For services requiring the customer to be at home, a widening of the time window is even

function $\sigma^{\text{FIXED}}(\bar{I}, \bigcup_{i\in\bar{I}}[c_i, d_i], D, j, [a, b])$
  if $\tau(\bar{I} \cup j, \bigcup_{i\in\bar{I}}[c_i, d_i] \cup \{[a, b]\}, D)$ **then**
    return $[a, b]$
  else
    return $[-\infty, \infty]$
  end if
end function

Algorithm A.2.1: Fixed policy

more problematic than a shift, as it forces the customer to take hours off. We therefore define a third level of service measure as the average amount of time window enlargement

$$\mathbb{L}^e = \frac{\sum_{i\in\bar{I}}(d_i - c_i) - (b_i - a_i)}{|\bar{I}|}.$$

A.2 DEFINING AND COMPUTING NEGOTIATION POLICIES

We propose online policies to support decisions of the provider during the online task of negotiating time windows with the customers. Formally, to define such an online decision policy corresponds to provide a definition for the $\sigma()$ procedure introduced in the previous Section.

We experimented on four policies. Each of them is based on the iterative checking of TSPTW feasibility problems: let $\tau()$ be a procedure taking in input a set of customers $\bar{I}$, their negotiated time windows $[c_i, d_i]$, their pairwise distances D_{ij} and their service time w_i, for each $i \in \bar{I}$, and giving as output a Boolean 'true' value if the resulting TSPTW problem has a feasible solution, 'false' otherwise, that is

$$\tau(\bar{I}, \bigcup_{i\in\bar{I}}[c_i, d_i], D) \rightarrow \{true, false\}.$$

In the following subsections A.2.1, A.2.2, A.2.3 and A.2.4 we detail our online policies, while in subsection A.2.5 we discuss some implementation details, including a graph model and a combinatorial algorithm for the effective computation of function $\tau()$.

A.2.1 *Fixed*

We first formalize the most intuitive policy: as soon as a new customer arrives, we check if it is possible to service her and all customers previously accepted in their desired time window. If so, the customer is

function $\sigma^{\text{SHIFT(F)}}(\bar{I}, \bigcup_{i\in\bar{I}}[c_i, d_i], D, j, [a, b])$
  $s^f \leftarrow 0$ ▷ Forward shift
  while not $\tau(\bar{I}\cup j, \bigcup_{i\in\bar{I}}[c_i, d_i]\cup\{[a + s^f \cdot k, b + s^f \cdot k]\}, D)$ **do**
    if $b + s^f \cdot k \geqslant T$ **then**
      $s^f \leftarrow +\infty$
      break
    end if
    $s^f \leftarrow s^f + 1$
  end while
  if $s^f < +\infty$ **then** ▷ Fine strategy
    $S \leftarrow \{\max\{0, s^f - 1\} \leqslant s \leqslant s^f\}$
    $s^f \leftarrow \min\{s \in S : \tau(\bar{I}\cup j, \bigcup_{i\in\bar{I}}[c_i, d_i]\cup\{[a + s \cdot k, b + s \cdot k]\}, D) = \text{true}\}$
  end if
  if $s^f < +\infty$ **then**
    return $[a + s^f \cdot k, b + s^f \cdot k]$
  else
    return $[-\infty, +\infty]$
  end if
end function

Algorithm A.2.2: Shift policy with forward fine strategy.

accepted without any change in its desired time window. Otherwise, the customer is rejected. A formal description of the 'Fixed' policy is reported in Pseudocode A.2.1.

A.2.2 *Shift policy*

Second, we consider a 'Shift' policy, that aims to accept each new customer, provided that a suitable shift in his desired time window can be found, allowing the operator to visit him and all the previously accepted customers. No change in the time window width is performed.

We consider two strategies for performing the time shift: *forward* and *backward*, in which a certain value s is added (resp. subtracted) to both a_i and b_i, that makes feasible the TSPTW instance including all accepted customers and the new one; if no such a value can be found without exceeding the daily deadline T (resp. 0), the customer is rejected. We also consider the *bidirectional* strategy in which both forward and backward shifts are computed, and the one requiring minimum shift is retained.

For what concerns the choice of value s, we take into account two strategies: in the *coarse* strategy the value s is chosen as a multiple of a base constant k, while in the *fine* strategy s can take any value.

A formal description of policy 'Shift', with forward, backward and bidirectional fine strategy, is reported in Pseudocode A.2.2, A.2.3 and

function $\sigma^{\text{SHIFT(B)}}(\bar{I}, \bigcup_{i\in\bar{I}}[c_i, d_i], D, j, [a, b])$
  $s^b \leftarrow 0$ ▷ Backward shift
  while not $\tau(\bar{I}\cup j, \bigcup_{i\in\bar{I}}[c_i, d_i]\cup\{[a - s^b\cdot k, b - s^b\cdot k]\}, D)$ **do**
    if $a - s^b\cdot k \leqslant 0$ **then**
      $s^b \leftarrow +\infty$
      break
    end if
    $s^b \leftarrow s^b + 1$
  end while
  if $s^b < +\infty$ **then** ▷ Fine strategy
    $S \leftarrow \{s : \max\{0, s^b - 1\} \leqslant s \leqslant s^b\}$
    $s^b \leftarrow \min\{s \in S : \tau(\bar{I}\cup j, \bigcup_{i\in\bar{I}}[c_i, d_i]\cup\{[a - s\cdot k, b - s\cdot k]\}, D) = \text{true}\}$
  end if
  if $s^b < +\infty$ **then**
    return $[a - s^b\cdot k, b - s^b\cdot k]$
  else
    return $[-\infty, +\infty]$
  end if
end function

Algorithm A.2.3: Shift policy with backward fine strategy.

function $\sigma^{\text{SHIFT(FB)}}(\bar{I}, \bigcup_{i\in\bar{I}}[c_i, d_i], D, j, [a, b])$
  $[c^f, d^f] \leftarrow \sigma^{\text{shift(f)}}(\bar{I}, \bigcup_{i\in\bar{I}}[c_i, d_i], D, j, [a, b])$
  $[c^b, d^b] \leftarrow \sigma^{\text{shift(b)}}(\bar{I}, \bigcup_{i\in\bar{I}}[c_i, d_i], D, j, [a, b])$
  if $\frac{d^f+c^f}{2} - \frac{b+a}{2} < \frac{b+a}{2} - \frac{d^b+c^b}{2}$ **then**
    return $[c^f, d^f]$
  else
    return $[c^b, d^b]$
  end if
end function

Algorithm A.2.4: Shift policy with bidirectional strategy.

A.2.4, respectively. Pure coarse strategies are obtained by simply discarding the blocks marked as 'Fine strategy'. From an algorithmic point of view, we assume rational data, and we compute the minima indicated in the fine strategy blocks by iterative dicotomic search.

A.2.3 *Enlarge policy*

Third, we devised an 'Enlarge' policy, that aims to accept each new customer by possibly enlarging its desired time window.

Also in this case we consider two strategies: *forward* and *backward*, in which we add (resp. subtract) a certain value e to b_i (resp. a_i), that makes feasible the TSPTW instance including all accepted customers and the new one; if no such a value can be found without exceeding the daily deadline T (resp. 0), the customer is rejected. As before, we

```
function σ^ENLARGE(F)(Ī, ⋃_{i∈Ī}[c_i, d_i], D, j, [a, b])
    e^f ← 0                                              ▷ Forward enlargement
    while not τ(Ī ∪ j, ⋃_{i∈Ī}[c_i, d_i] ∪ {[a, b + e^f · k]}, D) do
        if b + e^f · k ⩾ T then
            e^f ← +∞
            break
        end if
        e^f ← e^f + 1
    end while
    if e^f < +∞ then                                     ▷ Fine strategy
        E ← {e : (a − b)/k ⩽ e ⩽ e^f}
        e^f ← min{e ∈ E : τ(Ī ∪ j, ⋃_{i∈Ī}[c_i, d_i] ∪ {[a, b + e · k]}, D) = true}
    end if
    if e^f < +∞ then
        return [a, b + e^f · k]
    else
        return [−∞, +∞]
    end if
end function
```

Algorithm A.2.5: Enlarge policy with forward fine strategy.

analyze a third *bidirectional* strategy, that computes both forward and backward and retains the best.

Similarly to the 'Shift' policy, we take into account two ways of choosing e: in the *coarse* strategy the value e is chosen as a multiple of a base constant k, while in the *fine* strategy e can take any value.

A formal description of policy 'Enlarge' with forward, backward and bidirectional strategy is reported in Pseudocode A.2.5, A.2.6 and A.2.7, respectively; the minima indicated in the fine strategy blocks are also computed by iterative dicotomic search.

A.2.4 *Bucket policy*

Finally, we simulated a policy which is often used by industrial home service providers. We define q time *buckets*, that is a splitting of the daily working time $[0, T]$ in equal parts $[\ell \cdot T/q, (\ell + 1) \cdot T/q]$ for $\ell = 0 \dots (q - 1)$. These may correspond, for instance, to worker shifts, or to day fractions that are easy to visualize for the customers, like early-morning, late-morning, early-afternoon, late-afternoon. Then we replace the desired time window of each customer with that of the best fitting bucket that allows the operator to visit all the accepted customers and the new one; if no such a bucket can be found, the new customer is rejected. In our case, the regret of a bucket is computed as the difference between the central instant of the bucket and the central

function $\sigma^{\text{ENLARGE(B)}}(\bar{I}, \bigcup_{i\in\bar{I}}[c_i, d_i], D, j, [a, b])$
  $e^b \leftarrow 0$ ▷ Backward enlargement
  while not $\tau(\bar{I}\cup j, \bigcup_{i\in\bar{I}}[c_i, d_i]\cup\{[a - e^b \cdot k, b]\}, D)$ **do**
    if $a - e^b \cdot k \leqslant 0$ **then**
      $e^b \leftarrow +\infty$
      break
    end if
    $e^b \leftarrow e^b + 1$
  end while
  if $e^b < +\infty$ **then** ▷ Fine strategy
    $E \leftarrow \{e : (a - b)/k \leqslant e \leqslant e^b\}$
    $e^b \leftarrow \min\{e \in E : \tau(\bar{I}\cup j, \bigcup_{i\in\bar{I}}[c_i, d_i]\cup\{[a - e \cdot k, b]\}, D) = \text{true}\}$
  end if
  if $e^f < +\infty$ **then**
    return $[a - e^b \cdot k, b]$
  else
    return $[-\infty, +\infty]$
  end if
end function

Algorithm A.2.6: Enlarge policy with backward fine strategy.

function $\sigma^{\text{ENLARGE}}(\bar{I}, \bigcup_{i\in\bar{I}}[c_i, d_i], D, j, [a, b])$
  $[c^f, d^f] \leftarrow \sigma^{\text{enlarge(f)}}(\bar{I}, \bigcup_{i\in\bar{I}}[c_i, d_i], D, j, [a, b])$
  $[c^b, d^b] \leftarrow \sigma^{\text{enlarge(b)}}(\bar{I}, \bigcup_{i\in\bar{I}}[c_i, d_i], D, j, [a, b])$
  if $d^f - c^f < d^b - c^b$ **then**
    return $[c^f, d^f]$
  else
    return $[c^b, d^b]$
  end if
end function

Algorithm A.2.7: Enlarge policy with bidirectional strategy.

instant of the desired customer time window, being best those buckets having minimum regret. A formal description of policy 'Bucket' is reported in Pseudocode A.2.8.

A.2.5 *Implementation*

Three main issues arise in the implementation of our policies.

The first one is efficiently computing the procedure $\tau()$ to solve TSPTW feasibility problems. We embed a combinatorial algorithm based on dynamic programming, that is adapted from [39], briefly summarized in the following. We build a graph $G(V\cup\{p\}, E)$, where V is a set of vertices including one element v_i for each $i \in I$, p is an additional 'depot' vertex, and E is a set of edges, each having a weight equal to the time D_{ij} required to travel from customer i to customer j, or weight 0 if either i or j is the depot.

function $\sigma^{\text{BUCKET}}(\bar{I}, \bigcup_{i\in\bar{I}}[c_i, d_i], D, j, [a, b])$
  $Q \leftarrow \{0 \dots q-1\}$
  do
    if $Q = \emptyset$ **then**
      return $[-\infty, +\infty]$
    else
      $\ell^b \leftarrow \text{argmin}_{\ell\in Q}\{\| \frac{a-b}{2} - (\ell \cdot \frac{T}{q} + \frac{T}{2\cdot q})\|\}$
      $Q \leftarrow Q \setminus \{\ell^b\}$
    end if
  while $\tau(\bar{I} \cup j, \bigcup_{i\in\bar{I}}[c_i, d_i] \cup \{[\ell^b \cdot T/q, (\ell^b + 1) \cdot T/q]\}, D)$
  return $[\ell^b \cdot T/q, (\ell^b + 1) \cdot T/q]$
end function

Algorithm A.2.8: Bucket policy.

GREEDY CHECK. First, we run a nearest neighbor heuristic: we start from the depot at time 0 and iteratively move to the nearest unvisited node, until either all nodes are visited, or no node can be reached without exceeding the deadline T. If this simple check is enough to produce a feasible TSPTW solution, then we immediately stop, returning 'true' as $\tau()$ value.

LABEL STRUCTURE. Otherwise, we encode partial paths using *labels* $\lambda = (S, t, i)$, that store the set of visited vertices $S \subseteq V$, the overall time spent $0 \leqslant t \leqslant T$, and the last visited node $i \in V$.

INITIALIZATION. We create an initial label $\lambda = (\emptyset, 0, p)$, encoding the status of the operator waiting at the depot at the beginning of the day, and we mark it as *open*.

EXTENSION. Iteratively, we select the open label $\lambda' = (S', t', i')$ having minimum t' value, and we extend it to all other vertices $i'' \in V \cap S'$ corresponding to the unvisited customers that can be reached within their time windows, that is those for which the following inequality holds:

$$t' + D_{i'i''} + w_{i''} \leqslant b_{i''}.$$

For each of them, we create a new label $\lambda'' = (S' \cup \{i''\}, \max\{a_{i''}, t' + D_{i'i''}\} + w_{i''}, i'')$, that is marked as open. Label λ', instead, is marked as *closed*.

DOMINANCE RULES. After the extension, if two distinct labels $\lambda' = (S', t', i)$ and $\lambda'' = (S'', t'', i)$ are found, such that $S' \subseteq S''$ and

$t' \geqslant t''$, then label λ' is discarded, as any solution obtained by further extending λ' can also be obtained by extending λ'', obtaining a partial path visiting at least the same nodes and having at least the same available time.

TERMINATING CONDITION. The dynamic programming procedure stops in two cases: if a label is found to have $S = V$, then a feasible TSPTW solution has been found, and in this case the $\tau()$ result is 'true'; on the contrary, if there is no further open label, we certify that no feasible TSPTW solution can be found, and the $\tau()$ result is 'false'. If none of these conditions hold, we iterate the Extension and Dominance phases.

HEURISTIC IMPROVEMENT. We point out that, due to the online nature of the problem, it might be worth to *skip* a single customer, whose service is particularly involved, even if a feasible solution including her can be found, in order to be able to accept more customers appearing later. We actually found out to be beneficial to stop the dynamic programming algorithm in any case after that a certain number Δ of labels has been created, and to return a 'false' $\tau()$ value whenever this limit is reached; this showed to both dramatically speed up the computation and help in skipping those customers requiring long detours for servicing.

We remark that, even if much more sophisticated methods are proposed in the literature for the TSPTW [10], this simple algorithm is better suited when only TSPTW *feasibility* is concerned, like in the search for violated cuts in Vehicle Routing algorithms [49, 20].

The second implementation issue is the computation of minima for s and e values in the 'Shift' and 'Enlarge' policies. Given the effectiveness of the TSPTW feasibility dynamic programming algorithm, and assuming rational data, we decided to simply resort to an iterative dicotomic search. A range of possible feasible values is considered, as indicated in the set definition, and a tentative value is set as the mid-range of such possible feasible values. Then, the $\tau()$ function is computed on the resulting instance: if such a function returns 'true', then the tentative value is an upper bound on possible feasible values; otherwise, it is a lower bound. In any case the range of possible feasible values is halved. The dicotomic search stops as soon as lower and

upper bounds match.

The third implementation issue is choosing a suitable value for the base constant k. In an effort for balancing accuracy with speed, after a preliminary round of experiments we decided to set k to different values during the computation of $\sigma()$ for different customers. In particular, we obtained good results by setting it, for each customer, to half of the width of the desired service time window.

We finally remark that all our negotiation policies have a straightforward human-like interpretation. The technical computing-intensive details are hidden in the internal computation of function $\tau()$, that is on the other side what requires specialized combinatorial algorithms to be employed.

A.3 RESULTS AND DISCUSSION

We implemented our algorithms in C, using gcc 4.7 as a compiler, and running a set of experiments on a PC equipped with a 2.7 GHz CPU and 2 GB of RAM, under linux operating system.

As a benchmark we considered the set of instances of Pesant et al. [58], that were originally provided by Potvin and Bengio [61], and drawn from Solomon's dataset [71]. The benchmark consists of 30 feasible TSPTW instances involving up to 44 customers, that include a single depot. Indeed, besides being used also in recent publications [50], the size and feature of these instances well represent those of realistic home service delivery problems.

In order to check the behavior of our policies as the requests of the customers become more and more tight, we created three scenarii, indicated as datasets A, B, and C in the following, obtained by reducing each original time window $[a'_i, b'_i]$ by 25%, 50% and 75%, that is by setting

$$\begin{aligned} \alpha_i &= \frac{a'_i + b'_i}{2} \\ \beta_i &= b'_i - a'_i \\ a_i &= \alpha_i - r \cdot \frac{\beta_i}{2} \\ b_i &= \alpha_i + r \cdot \frac{\beta_i}{2} \end{aligned}$$

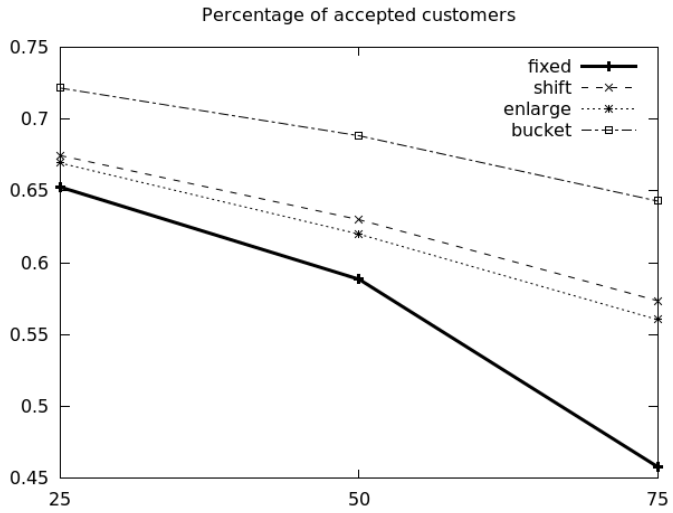

Figure 15: Average $\mathbb{L}^a$ values of acceptance policies as the desired service time windows reduce.

for r equal to 0.75, 0.50 and 0.25, respectively. Therefore, our overall testbed consists of 90 instances. Each service time w_i was set to 15; the availability time window of the depot has been left unchanged.

We report our results in Table 17, Table 18 and Table 19 for datasets A, B and C, respectively. Each table contains one row for each combination of policies and computing strategy (as indicated in the first two columns), and reports in turn the percentage of accepted customers $\mathbb{L}^a$ (column 'Acc. cust.'), the amount of average shift $\mathbb{L}^s$ and enlarge $\mathbb{L}^e$ values among accepted customer (columns 'Rel. shift' and 'Rel. enlarge'), and the average CPU time required to negotiate each service time window (column 'CPU time'), that is in a practical setting the average *query* time, in which a customer must wait between her request and a confirmation, or an offer for an alternative service time window.

In a preliminary round of experiments we found that fixing a maximum number of labels $\Delta = 2000$ in the dynamic programming procedure, gave a good compromise between solution quality and CPU time. Indeed, as can be observed by looking at the 'CPU time' column of the tables, the average query time is always less than a tenth of a second, matching our proposal of producing a real-time tool.

A.3.1 *Evaluation of* $\mathbb{L}^a$ *measure.*

As a first test, we analyzed the number of customers that can be accepted in their desired day by employing different policies; that is, we

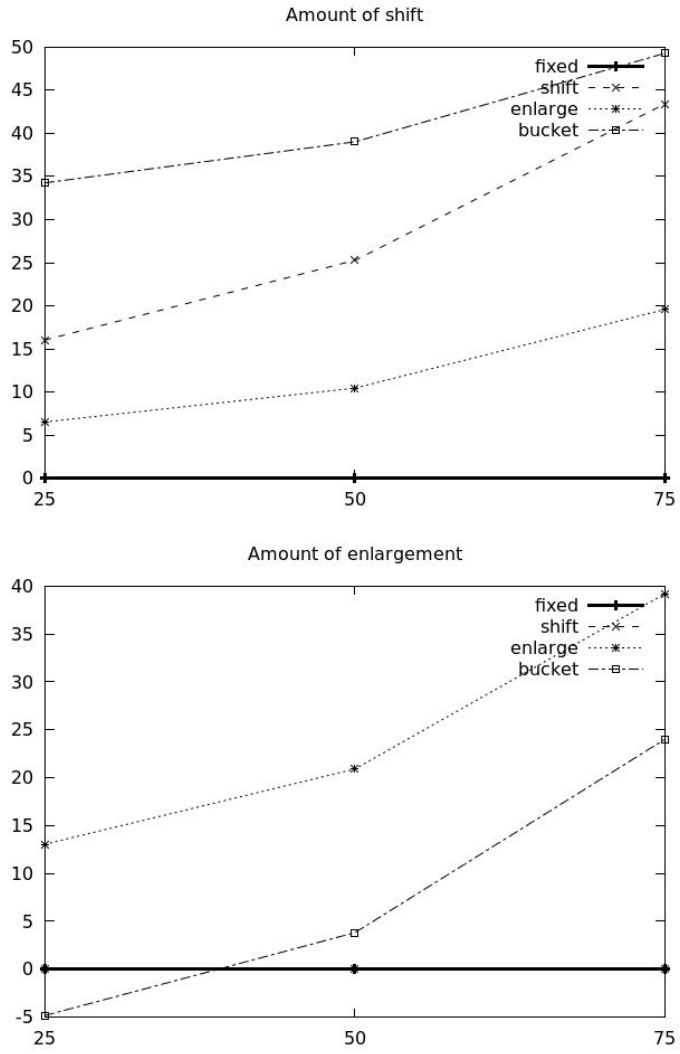

Figure 16: Average $\mathbb{L}^s$ (left) and $\mathbb{L}^e$ (right) values of acceptance policies as the desired service time windows reduce.

compare our policies with respect to measure $\mathbb{L}^a$. The results reported in the 'Acc. cust.' columns of tables 17, 18 and 19 are further aggregated in Figure 15. It includes one line for each policy. The 'bucket' policy performs best, yielding from 6.5% to almost 20% improvement with respect to the 'fixed' policy. Policies 'shift' and 'enlarge' perform similarly: they offer a few percentage points improvement with respect to 'fixed' in dataset A, but such an improvement increases as the desired time windows reduce, reaching more than 10% on dataset C.

A.3.2 *Evaluation of $\mathbb{L}^s$ and $\mathbb{L}^e$ measures.*

Then we compared our acceptance policies with respect to the amount of shift and enlargement they yield on accepted customers. As in the previous section, we further aggregate the results reported in 'Rel. shift' and 'Rel. enlarge' columns in tables 17, 18 and 19 in a single Figure 16 for each policy, that reports time windows reduction values on the horizontal axis and average $\mathbb{L}^s$ and $\mathbb{L}^e$ values on the vertical axes, in order to highlight the quality measure trend as the customer desired time windows reduce. Measures worsen mildly as the time windows reduce. No policy eventually overtakes the others. While 'shift' policy produces by construction solutions having zero $\mathbb{L}^e$ values, the 'enlargement' policy is actually able to reduce shift amount by introducing enlargement. By using 'buckets' policy, in dataset A and dataset B it is even possible to give to the customer a service time window which is narrower than the desired one, at the expense of higher shift values.

A.3.3 *Comparison of policies and strategies.*

Finally, we tried to perform an overall assessment on our acceptance policies and strategies.

First, we compared how sensitive each policy is with respect to changes in its computing strategy. By looking at the difference between the best and worst average result in the tables, for each dataset, the less stable policy in terms of both measure $\mathbb{L}^a$ and $\mathbb{L}^s$ shows to be 'shift'; 'bucket' is in general the most stable, but 'enlarge' offers comparable results.

Then, in Figure 17 we present a scatter plot for instances in dataset A (top), dataset B (middle) and dataset C (bottom). Each point in the plots represents a single run of our experiments, that is the application of a particular acceptance policy to a particular instance. Runs performed by applying 'shift', 'enlargement' and 'bucket' policies are marked with squares, triangles and circles, respectively. Each plot reports $\mathbb{L}^s$ values on the vertical axis and $\mathbb{L}^e$ values on the horizontal one: the coordinates of each point represent the $\mathbb{L}^s$ and $\mathbb{L}^e$ values obtained in the corresponding run; instead, the size of each point is proportional to the $\mathbb{L}^a$ value obtained.

No dominance can be observed among the acceptance policies. The 'shift' and 'enlarge' policies tend to produce more clustered, and therefore predictable, results; 'bucket' results are more spread.

For what concerns the strategy selection, by looking at tables 17, 18 and 19 two settings seem to be particularly appealing for both 'shift' and 'enlarge': bidirectional coarse and backward fine. In fact, the former strategy seems to consistently yield a higher fraction of accepted customers $\mathbb{L}^a$ at the expense of higher $\mathbb{L}^s$ and $\mathbb{L}^e$ values; the latter produces good solutions in terms of $\mathbb{L}^s$ and $\mathbb{L}^e$ measures by discarding a few more customers. This behavior can be expected, as the former strategy follows a rationale of accommodating customers without over-optimizing their negotiated time windows, while the latter tries to provide tight negotiated time windows, that in turn put hard constraints on the operator schedule, and eventually prevents new customers to be accepted. Backward strategy has some advantage over forward strategy; indeed, using time later in the working day tends in general to constrain more the schedule of the operator. For the 'bucket' policy, the strategy using 8 slots seems to provide the best overall behavior, accepting more customers with a modest increase in $\mathbb{L}^s$ values and a strong reduction in $\mathbb{L}^e$ values.

We therefore compared the average behavior of these policy-strategy combination. A synthesis of this comparison is depicted in Figure 18. These charts have the same structure of those in figures 15 and 16, and contain one line for each of the following policy-strategy combination: fixed, shift bidirectional coarse, shift backward fine, enlarge bidirectional coarse, enlarge backward fine, bucket 8 slots.

A.4 CONCLUSIONS

We tackled a key slot allocation problem arising in home service delivery, modeling it as an online optimization problem to be solved in realtime, and proposing evaluation measures to assess the quality of the solutions produced. We designed strategies to be applied by a service provider that needs to match customer requests and operator's time availability; even if their computation require combinatorial optimization algorithms to be run, their behavior is still of easy interpretation for human operators. We performed extensive computational evaluation of these policies. As a main result, we found out that the fraction of accepted customers can be substantially increased by allowing for reasonable changes in their desired time window. For what concerns

efficiency, our algorithmic tools prove to be suitable to be embedded in a real-time decision support system.

Policy	Parameters	Acc. cust. $\mathbb{L}^a$ (%)	Rel. shift $\mathbb{L}^s$	Rel. enlarge $\mathbb{L}^e$	CPU time (s)
fixed		65.26	0.00	0.00	0.004
shift	bidirectional, coarse	69.73	23.81	0.00	0.004
	bidirectional, fine	67.70	17.42	0.00	0.004
	backward, coarse	67.04	8.54	0.00	0.004
	backward, fine	65.45	8.31	0.00	0.004
	forward, coarse	68.85	23.09	0.00	0.004
	forward, fine	65.98	15.02	0.00	0.004
enl	bidirectional, coarse	68.71	8.88	17.76	0.004
	bidirectional, fine	67.15	7.61	15.26	0.004
	backward, coarse	67.10	4.68	9.36	0.004
	backward, fine	65.27	3.44	6.90	0.004
	forward, coarse	67.34	7.90	15.80	0.004
	forward, fine	66.05	6.63	13.28	0.004
bucket	4	73.06	32.20	2.69	0.004
	6	71.52	38.14	-6.33	0.004
	8	71.88	32.46	-10.84	0.004

Table 17: Results on Dataset A (25% size reduction).

Policy	Parameters	Acc. cust. $\mathbb{L}^a$ (%)	Rel. shift $\mathbb{L}^s$	Rel. enlarge $\mathbb{L}^e$	CPU time (s)
fixed		58.83	0.00	0.00	0.002
shift	bidirectional, coarse	65.88	34.00	0.00	0.002
	bidirectional, fine	63.51	24.12	0.00	0.002
	backward, coarse	61.54	18.33	0.00	0.002
	backward, fine	60.09	17.04	0.00	0.002
	forward, coarse	63.40	30.00	0.00	0.002
	forward, fine	63.52	28.18	0.00	0.002
enl	bidirectional, coarse	63.84	11.75	23.50	0.002
	bidirectional, fine	62.27	9.44	18.95	0.002
	backward, coarse	61.09	8.64	17.28	0.002
	backward, fine	59.75	7.67	15.39	0.002
	forward, coarse	62.51	12.88	25.76	0.002
	forward, fine	62.45	12.29	24.63	0.002
bucket	4	68.28	35.53	14.08	0.002
	6	69.01	39.93	2.62	0.002
	8	69.22	41.62	-5.22	0.002

Table 18: Results on Dataset B (50% size reduction).

Policy	Parameters	Acc. cust. $\mathbb{L}^a$ (%)	Rel. shift $\mathbb{L}^s$	Rel. enlarge $\mathbb{L}^e$	CPU time (s)
fixed		45.77	0.00	0.00	0.001
shift	bidirectional, coarse	64.66	54.85	0.00	0.001
	bidirectional, fine	57.95	31.69	0.00	0.001
	backward, coarse	57.12	40.77	0.00	0.001
	backward, fine	52.62	29.94	0.00	0.001
	forward, coarse	58.66	63.30	0.00	0.001
	forward, fine	53.07	39.95	0.00	0.001
enl	bidirectional, coarse	61.67	23.43	46.86	0.001
	bidirectional, fine	56.90	14.31	28.76	0.001
	backward, coarse	56.32	18.91	37.82	0.001
	backward, fine	52.05	14.59	29.31	0.001
	forward, coarse	56.13	27.19	54.38	0.001
	forward, fine	53.11	19.05	38.19	0.001
bucket	4	62.90	45.12	40.81	0.001
	6	64.52	49.96	21.96	0.001
	8	65.42	52.80	9.28	0.001

Table 19: Results on Dataset C (75% size reduction).

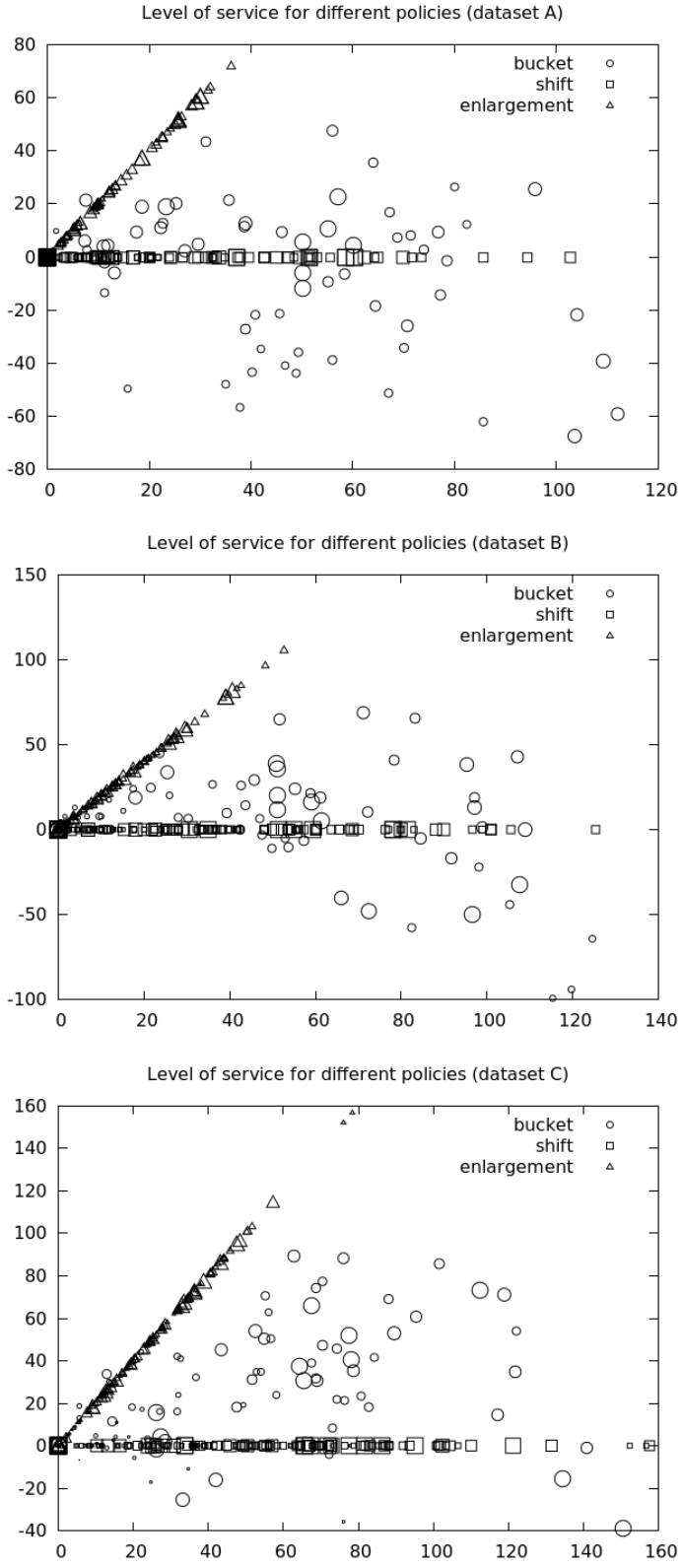

Figure 17: Comparison of level of service measures for acceptance policies on dataset A (top) dataset B (middle) and dataset C (bottom)

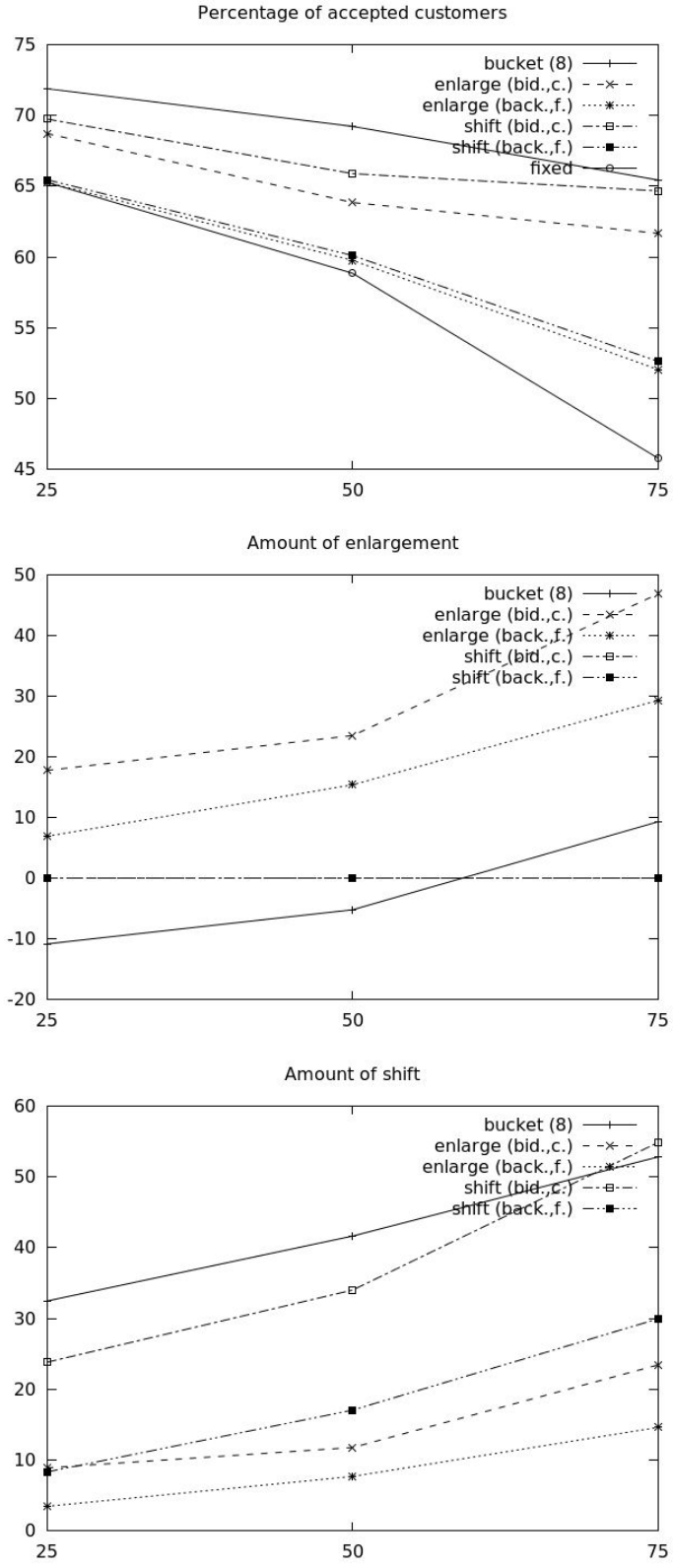

Figure 18: Behavior of the best performing acceptance policies and strategies as the desired service time windows reduce.

BIBLIOGRAPHY

[1] T. Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[2] N. Agatz, A. Campbell, M. Fleischmann, and M. Savelsbergh. Time slot management in attended home delivery. *Transportation Science*, 45(3):435–449, 2011.

[3] Niels Agatz, AnnMelissa Campbell, Moritz Fleischmann, and Martin Savels. Challenges and opportunities in attended home delivery. In Bruce Golden, S. Raghavan, and Edward Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces*, pages 379–396. Springer US, 2008.

[4] C. Archetti and M.G. Speranza. Vehicle routing problems with split deliveries. *International Transactions in Operational Research*, 19(1–2):3–22, 2012.

[5] Claudia Archetti and Maria Grazia Speranza. An overview on the split delivery vehicle routing problem. In Karl-Heinz Waldmann and Ulrike M. Stocker, editors, *Operations Research Proceedings 2006*, Operations Research Proceedings, pages 123–127. Springer Berlin Heidelberg, 2007.

[6] Claudia Archetti, Martin W. P. Savelsbergh, and M. Grazia Speranza. Worst-case analysis for split delivery vehicle routing problems. *Transportation Science*, 40(2):pp. 226–234, 2006.

[7] Claudia Archetti, Maria Garzia Speranza, and Alain Hertz. A Tabu Search Algorithm for the Split Delivery Vehicle Routing Problem. *Transportation Science*, 40(1):64–73, February 2006.

[8] Claudia Archetti, Nicola Bianchessi Maria Garzia Speranza, and Alain Hertz. A column generation approach for the split delivery vehicle routing problem. *NETWORKS*, 2011.

[9] Claudia Archetti, Nicola Bianchessi, and M. Grazia Speranza. Branch-and-cut algorithms for the split delivery vehicle routing problem. *European Journal of Operational Research*, 238(3):685–698, 2014.

[10] R. Baldacci, A. Mingozzi, and R. Roberti. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(3):356–371, 2012.

[11] Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385, 2008.

[12] Maria Battarra, Jean-François Cordeau, and Manuel Iori. Pickup and delivery problems for goods transportation. In Paolo Toth and Daniele Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, chapter 6, pages 161–191. SIAM, 2014.

[13] G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171 (1):85–106, 2006.

[14] Hatem Ben Amor, Jacques Desrosiers, and Jos Manuel Valrio de Carvalho. Dual-optimal inequalities for stabilized column generation. *Opereation Research*, 54(3):454–463, May 2006.

[15] Enrique Benavent, Mercedes Landete, Enrique Mota, and Gregorio Tirado. The multiple vehicle pickup and delivery problem with {LIFO} constraints. *European Journal of Operational Research*, 243(3):752–762, 2015.

[16] G. Berbeglia, J.-F. Cordeau, and G. Laporte. A hybrid tabu-search and constraint programming algorithm for the dynamic dial-a-ride problem. *INFORMS Journal on Computing*, 24:343–355, 2012.

[17] A.M. Campbell and M. Savelsbergh. Incentive schemes for attended home delivery services. *Transportation Science*, 40(3):327–341, 2006.

[18] Emmanuel J. Candès, Michael B. Wakin, and Stephen P. Boyd. Enhancing sparsity by reweighted ℓ_1 minimization. *Journal of Fourier Analysis and Applications*, 14(5-6):877–905, 2008.

[19] Alberto Ceselli, Giovanni Righini, and Matteo Salani. A column generation algorithm for a rich vehicle-routing problem. *Transportation Science*, 43(1):56–69, 2009.

[20] Alberto Ceselli, Giovanni Righini, and Emanuele Tresoldi. Combined location and routing problems for drug distribution. *Discrete Applied Mathematics*, 165:130–145, 2014.

[21] Alberto Ceselli, Angel Felipe, M. Teresa Ortuño, Giovanni Righini, and Gregorio Tirado. A branch-and-cut-and-price algorithm for the green vehicle routing problem with partial recharge and multiple technologies. In *ODYSSEUS 2015*, 2015.

[22] D. Chemla. *Algorithms for optimizing shared mobility systems*. Theses, Université Paris-Est, October 2012. URL https://pastel.archives-ouvertes.fr/pastel-00839521.

[23] Daniel Chemla, Frédéric Meunier, and Roberto Wolfler Calvo. Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization*, 10(2):120–146, 2013.

[24] Marilène Cherkesly, Guy Desaulniers, Stefan Irnich, and Gilbert Laporte. Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and multiple stacks. *European Journal of Operational Research*, 2015.

[25] F. Chung, R. Graham, J. Mao, and G. Varghese. Parallelism versus memory allocation in pipelined router forwarding engines. *Theory of Computer Systems*, 39(6):829–849, 2006.

[26] François Clautiaux, Cláudio Alves, José Valério de Carvalho, and Jürgen Rietz. New stabilization procedures for the cutting stock problem. *INFORMS Journal on Computing*, 23(4):530–545, 2011.

[27] Gianni Codato and Matteo Fischetti. Combinatorial benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4): 756–766, 2006.

[28] Claudio Contardo, Catherine Morency, and Louis-Martin Rousseau. Balancing a Dynamic Public Bike-Sharing System. Technical report, March 2012.

[29] J.F. Cordeau, G. Laporte, M.W.P. Savelsbergh, and D. Vigo. *Vehicle Routing*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 367–428. Elsevier, 2007.

[30] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8:101–111, 1960.

[31] George B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–288, 1957.

[32] Mauro Dell'Amico, Eleni Hadjicostantinou, Manuel Iori, and Stefano Novellani. The bike sharing rebalancing problem: Mathematical formulations and benchmark instances. *Omega*, 45:7–19, 2014.

[33] M. Delorme, M. Iori, and S. Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. 2014. URL http://or.dei.unibo.it/library/bpplib.

[34] P. DeMaio. Bike-sharing: History, impacts, models of provision, and future. *Journal of Public Transportation*, 12(4):41–56, 2009.

[35] Guy Desaulniers. Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows. *Operations Research*, 58(1):179–192, 2010.

[36] CPLEX development team. Ibm ilog cplex optimization studio: Cplex user's manual - version 12 release 4. Technical report, IBM corp., 2011.

[37] Karl F. Doerner and Juan-José Salazar-González. Pickup-and-delivery problems for people transportation. In Paolo Toth and Daniele Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, chapter 7, pages 193–212. SIAM, 2014.

[38] Moshe Dror and Pierre Trudeau. Split delivery routing. *Naval Research Logistics*, 37:383–402, 1990.

[39] Y. Dumas, J. Desrosiers, E. Gelinas, and M.M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43:367–371, 1995.

[40] L. Epstein and R. van Stee. Improved results for a memory allocation problem. *Theory of Computing Systems*, 48(1):79–92, 2011.

[41] L. Epstein, A. Levin, and R. van Stee. Approximation schemes for packing splittable items with cardinality constraints. *Algorithmica*, 62(1–2):102–129, 2012.

[42] Güneş Erdoğan, Gilbert Laporte, and Roberto Wolfler Calvo. The static bicycle relocation problem with demand intervals. *European Journal of Operational Research*, 238(2):451–457, 2014.

[43] Güneş Erdoğan, Maria Battarra, and Roberto Wolfler Calvo. An exact algorithm for the static rebalancing problem arising in bicycle sharing systems. *European Journal of Operational Research*, 245 (3):667–679, 2015.

[44] L. Grandinetti, F. Guerriero, F. Pezzella, and O. Pisacane. The multi-objective multi-vehicle pickup and delivery problem with time windows. *Procedia - Social and Behavioral Sciences*, 111:203–212, 2014. Transportation: Can we do more with less resources? - 16th Meeting of the Euro Working Group on Transportation - Porto 2013.

[45] T. Gschwind and S. Irnich. Dual inequalities for stabilized column generation revisited. Technical Report Working Paper n. 1407, Gutenberg School of Management and Economics, Mainz Univ., 2014.

[46] F. Hennig, B. Nygreen, M. Christiansen, K. Fagerholt, K.C. Furman, J. Song, G.R. Kocis, and P.H. Warrick. Maritime crude oil transportation – a split pickup and split delivery problem. *European Journal of Operational Research*, 218(3):764–774, 2012.

[47] Frank Hennig, Bjørn Nygreen, and Marco E. Lübbecke. Nested column generation applied to the crude oil tanker routing and scheduling problem with split pickup and split delivery, 2012.

[48] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[49] N. Kohl, J. Desrosiers, O.B.G. Madsen, M. M. Solomon, and F. Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116, 1999.

[50] M. Lòpez-Ibáñez, C. Blum, J. W. Ohlmann, and B. W. Thomas. The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. *Applied Soft Computing*, 13(9):3806–3815, 2013.

[51] G. Laporte. Scheduling issues in vehicle routing. *Annals of Operations Research*, in press, 2013.

[52] Neal Lathia, Saniul Ahmad, and Licia Capra. Measuring the impact of opening the london shared bicycle scheme to casual users. *Transportation Research Part C*, 22:88–102, 2012.

[53] Jenn-Rong Lin and Ta-Hui Yang. Strategic design of public bicycle sharing systems with service level constraints. *Transportation Research Part E: Logistics and Transportation Review*, 47(2):284–294, 2011.

[54] Andrea Lodi, Silvano Martello, Michele Monaci, Claudio Cicconetti, Luciano Lenzini, Enzo Mingozzi, Carl Eklund, and Jani Moilanen. Efficient two-dimensional packing algorithms for mobile wimax. *Management Science*, 57(12):2130–2144, 2011.

[55] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990. ISBN 0-471-92420-2.

[56] N. Menakerman and R. Rom. Bin packing with item fragmentation. In *Proceedings of the 7th International Workshop on Algorithms and Data Structures, in: Lecture Notes in Computer Science*, pages 313–324, 2001.

[57] Christoph Nitsche, Guntram Scheithauer, and Johannes Terno. Tighter relaxations for the cutting stock problem. *European Journal of Operational Research*, 112(3):654–663, 1999.

[58] G. Pesant, M. Gendreau, J.-Y. Potvin, and J.-M. Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32 (1):12–29, 1998.

[59] Bjørn Petersen and David Pisinger. *Shortest Paths and Vehicle Routing*. PhD thesis, 2011.

[60] V. Pillac, M. Gendreau, C. Guéret, and A.L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

[61] J.-Y. Potvin and S. Bengio. The vehicle routing problem with time windows part ii: Genetic search. *INFORMS Journal on Computing*, 8:165–172, 1996.

[62] Luigi Di Puglia Pugliese and Francesca Guerriero. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013.

[63] Tal Raviv and Ofer Kolka. Optimal inventory management of a bike-sharing station. *IEEE Transactions*, to appear.

[64] Tal Raviv, Michal Tzur, and Iris A. Forma. Static repositioning in a bike-sharing system: models and solution approaches. *EURO Journal on Transportation and Logistics*, 2(3):187–229, 2013.

[65] Giovanni Righini and Matteo Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3 (3):255–273, 2006.

[66] M. Schneider, A. Stenger, and D. Goeke. The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, 48(4):500–520, 2014.

[67] Jasper Schuijbroek, Robert Hampshire, and Willem-Jan van Hoeve. Inventory rebalancing and vehicle routing in bike sharing systems. Technical report, 2013.

[68] S. Secci, A. Ceselli, F. Malucelli, A. Pattavina, and B. Sansò. Direct optimal design of a quasi-regular composite-star core network. In *IEEE Proc. of DRCN*, pages 7–10, 2007.

[69] H. Shachnai, T. Tamir, and O. Yehezkely. Approximation schemes for packing with item fragmentation. In Thomas Erlebach and Giuseppe Persinao, editors, *Approximation and Online Algorithms*, volume 3879 of *Lecture Notes in Computer Science*, pages 334–347. Springer Berlin / Heidelberg, 2006.

[70] N. Skorin-Kapov. Routing and wavelength assignment in optical networks using bin packing based algorithms. *European Journal of Operational Research*, 177(2):1167–1179, 2007.

[71] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time windows. *Operations Research*, 35:254–265, 1987.

[72] R. Spliet and G. Desaulniers. The discrete time window assignment vehicle routing problem. Technical report, HEC, 2012.

[73] Magnus Stålhane, Henrik Andersson, Marielle Christiansen, Jean-François Cordeau, and Guy Desaulniers. A branch-price-and-cut method for a ship routing and scheduling problem with split loads. *Computers & Operations Research*, 39(12):3361–3375, 2012.

[74] S. Utz, U. Matzat, and C. Snijders. On-line reputation systems: The effects of feedback comments and reactions on building and rebuilding trust in on-line auctions. *International Journal of Electronic Commerce*, 13(3):95–118, 2009.

[75] J.M. Valério de Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86(0):629–659, 1999.

[76] José M. Valério de Carvalho. Using extra dual cuts to accelerate column generation. *INFORMS Journal on Computing*, 17(2):175–182, 2005.

[77] J.M. Valério de Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86(0):629–659, 1999.

[78] José M. Valério de Carvalho. Using extra dual cuts to accelerate column generation. *INFORMS J. on Computing*, 17(2):175–182, April 2005.

[79] François Vanderbeck. A nested decomposition approach to a three-stage, two-dimensional cutting-stock problem. *Manage. Sci.*, 47(6):864–879, June 2001.

[80] François Vanderbeck. Implementing mixed integer column generation. In Guy Desaulniers, Jacques Desrosiers, and MariusM. Solomon, editors, *Column Generation*, pages 331–358. Springer US, 2005. ISBN 978-0-387-25485-2.

[81] Patrick Vogel and Dirk C. Mattfeld. Anticipating usage patterns in the design of bike-sharing systems. In *INFORMS 2011; Shared Mobility Systems*, 2011.

[82] Y. Xing, L. Li, Z. Bi, M. Wilamowska-Korsak, and L. Zhang. Operations research (or) in service industries: A comprehensive review. *Systems Research and Behavioral Science*, 30(3):300–353, 2013.

[83] Yun-Bin Zhao and Duan Li. Reweighted ℓ_1-minimization for sparse solutions to underdetermined linear systems. *SIAM Journal on Optimization*, 22(3):1065–1088, 2012.