Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

The Multiple Multidimensional Knapsack with Family-Split Penalties

Simona Mancini[a], Michele Ciavotta[b], Carlo Meloni[c,*]

[a] Dipartimento di Matematica ed Informatica, Università di Cagliari, Cagliari, Italy
[b] Dipartimento di Informatica, Sistemistica e Comunicazione, Università di Milano-Bicocca, Milano, Italy
[c] Dipartimento di Ingegneria Elettrica e dell'Informazione, Politecnico di Bari, Bari, Italy

ARTICLE INFO

Article history:
Received 28 November 2018
Accepted 23 July 2019
Available online 27 July 2019

Keywords:
Combinatorial optimization
Knapsack Problems
Integer programming
Benders' cuts
Resource assignment

ABSTRACT

The Multiple Multidimensional Knapsack Problem with Family-Split Penalties (MMdKFSP) is introduced as a new variant of both the more classical Multi-Knapsack and Multidimensional Knapsack Problems. It reckons with items categorized into families and where if an individual item is selected to maximize the profit, all the items of the same family must be selected as well. Items belonging to the same family can be assigned to different knapsacks; however, in this case, split penalties are incurred. This problem arises in resource management of distributed computing contexts and Service Oriented Architecture environments. An exact algorithm based on the exploitation of a specific combinatorial Benders' cuts approach is proposed. Computational experiments on different sets of benchmark test problems show the effectiveness of the proposed algorithm. The comparison against a state-of-the-art commercial solver confirms the validity of the proposed approach considering also the scalability issue.

1. Introduction

Knapsack Problems are among the most studied and challenging classes of problems in combinatorial optimization (Kellerer, Pferschy, & Pisinger, 2004; Martello & Toth, 1990). In its standard version, the Knapsack Problem (KP) considers: (i) n items available to be selected, each of them (indicated as $i \in \{1, \dots, n\}$) featuring a value of p_i, i.e., a measure of an utility or a profit of item i, as well as a size w_i; (ii) a natural constraint assuring that the aggregated size of all selected items cannot exceed the capacity c of the knapsack; (iii) the objective the decision-maker has to achieve by selecting a subset of the items is the optimization of the overall knapsack appraisal (defined for instance as the sum of the values of the selected items) under the capacity constraint described in the previous point (ii).

Despite having a pretty straightforward definition, the standard KP encompasses a class of non-trivial integer combinatorial optimization problems that are classified as $\mathcal{NP}$-hard (Garey & Johnson, 1979). Moreover, even the standard KP is still of great interest due to its applicability in many areas such as business (Melachrinoudis & Kozanidis, 2002; Pfeiffer, 2007), networking (Ceri, Martella, & Pelagatti, 1982; Wang, Yang, Zhang, Wu, & Xu, 2007), scheduling (Brucker & Knust, 2011), logistics (Bramel & Simchi-Levi, 1997; Kim & Günther, 2007), and even cryptography (Chor & Rivest, 1988).

Over the years, the *original* knapsack formulation has been extended and several *variants* have been discussed in the literature. A considerable number of them have been obtained by changing the value of some problem parameters such as the number of items to select, the number of objectives, or even the number of knapsacks (Kellerer et al., 2004; Martello & Toth, 1990). Particularly, considering more than one knapsack has led to the definition of the Multiple Knapsack Problem (MKP) that, instead of a single knapsack, considers multiple knapsacks, each one, say j, having its own capacity c_j, $j = 1, \dots, m$ (being m the number of available knapsacks). The decision-maker here is not only interested in the selection of a single item but also in which knapsack allocate it; this non-trivial variation leads to the formulation of a class of problems arising, among others, in logistics and scheduling (Bramel & Simchi-Levi, 1997). A further extension of the classical KP is the so called Multidimensional Knapsack Problem (MdKP) (Fréville, 2004), in which a full set of attributes is associated with each item (e.g., volume, weight, etc.), whereas the original capacity constraint is replaced by a set of constraints on the maximum amount of each attribute value sustainable by each knapsack, making the problem also multi-constrained. This paper introduces a novel variant of both the more classical MKP and

* Corresponding author.
E-mail addresses: simona.mancini@polito.it (S. Mancini), michele.ciavotta@unimib.it (M. Ciavotta), carlo.meloni@poliba.it (C. Meloni).

https://doi.org/10.1016/j.ejor.2019.07.052
0377-2217/© 2019 Elsevier B.V. All rights reserved.

MdKP, namely the Multiple Multidimensional Knapsack Problem with Family-Split Penalties (MMdKFSP), that often arises in resource management of distributed computing contexts and Service Oriented Architecture (SOA) environments. In this problem, the items are categorized into *families* of strongly related units. Such a forceful belonging is conveyed in the problem by the condition that if one family member is selected then all the other components of the same family must be selected as well. Moreover, although items of the same *group* can be assigned to different knapsacks, if this occurs, family-split *penalties* are applied; thus affecting the overall profit.

The main contributions of this paper can be summarized as follows. Besides the introduction of the MMdKFSP, specified by means of a Integer Linear Programming (ILP) formulation, the paper reports on a research activity intended to propose and evaluate an *exact* solution approach for the problem. More in detail, an exact algorithm for the MMdKFSP is proposed based on the application of a specific Combinatorial Benders' Cuts (CBC) approach characterized by a problem re-formulation, and two different types of cuts. A specific pre-processing procedure completes the algorithmic scheme that overall deviates significantly from the standard CBC procedure (Benders, 1962). Computational experiments on different sets of benchmark test instances have been conducted and their results demonstrated the soundness and the effectiveness of the proposed solution with respect to a baseline represented by an ILP formulation solved by a state-of-the-art commercial solver considering also the scalability issue. The instances generated to test the proposed algorithm are available online for further studies and researches.

This paper is organized as follows. Section 2 highlights the motivations behind this research work. Section 3 is devoted to the review of the related relevant literature. In Section 4 the formal description of the problem is provided including an integer programming formulation. The proposed solution method and the implementation details are illustrated in Section 5. The experimental validation is described and discussed in Section 6. Conclusions follow in Section 7 reporting also indications of possible future research directions.

2. Motivation

The MMdKFSP is a *novel* Knapsack Problem variant that is proposed for the first time in this paper. It arises, for instance, in resource management of distributed computing applications and SOA environments both on a departmental or local scale, and on a larger scale (e.g., public cloud environments). Interestingly, this problem does not fall into already studied knapsack-like cases, and considering its relevance, it deserves to be addressed in order to find appropriate solution approaches for applications and, certainly, a first significant step in this direction includes the design and evaluation of exact optimization algorithms.

Regarding the structure of the problem, in the considered application context, items usually represent computing tasks or services (often in form of web services within Virtual Machines or Containers, da Silva, Kirikova, & Alksnis, 2018; Mazumdar & Pranzo, 2017), with different requirements on the available resources, that are categorized into families according to software application, ownership, or complementarity of requirements. Considering the scenario where multiple web applications have to be assigned to (possibly scarce) computing resources, a multi-tier web application in a distributed environment is generally constituted by several collaborating (often replicated) long-running software agents communicating via application-scoped virtual networks. The application can work properly only if all its components are in execution and accessing the application networks; from this derives the concept of *all-or-nothing* families of items. Moreover, although for availability reasons the best practices of cloud computing demand to distribute the replicas of the application components across different *zones* (Varia, 2010) (e.g., Amazon[1] or Azure Availability zones[2]) of the same data center, it is advisable to place at least one replica of each component in the same zone to exploit high-speed connections among them. Furthermore, major cloud vendors offers dedicated deployments (as Amazon Dedicated Instances[3] and Dedicated Hosts[4]) where the components of applications belonging to the same user/organization are physically isolated from those of other accounts. In addition, Software as a Service (SaaS) applications are increasingly made up of containerized web services (Truyen et al., 2016) deployed on virtual machines provisioned on a Cloud provider (e.g., Amazon Elastic Container Service for Kubernetes (EKS)); also in this case allocating all the tiers on a single machine, potentially replicating them on a different one, might be beneficial in terms of memory-speed connections and reduced operations activities and higher availability. Note that, in the case of fully replicated applications each deployment can be considered as a family for the purposes of the problem.

Lastly, in smaller-sized scenarios, such as the case of a small-medium organization, where virtualization and automation are not fully implemented, applications' components tend to be deployed on the bare-metal, bundled as much as possible together to avoid too complex operations activities like the configuration of virtual networks distributed across different physical machines and switches (software defined networks, Chowdhury & Boutaba, 2010). In similar situations, replication for availability and for variable-workload handling is generally avoided since the workload is known in advance and stable (thus the components can be equipped with the right amount of resources to support it) whereas downtimes can be tolerated with the goal of maximizing the number of running applications; moreover, rarely the applications are mission-critical in such circumstances.

The optimization model formalized and exactly solved in this work has to be considered as a first step towards the creation of an optimal resource-aware deployment planner for component-based distributed applications. In such scenario, knapsacks represent machines (e.g., VMs, physical servers and computers) characterized by a limited availability of different resources (e.g., CPU, RAM, storage, networking, etc.). Items, i.e., the components of the applications waiting to be placed and executed on the machines, have their requirements in terms of use or occupation of resources, and when one item is selected in order to maximize an overall performance index, its family must be completely selected as well, i.e. a family cannot be partially selected because an application might not be functional without some of its parts. However, items belonging to the same family can be assigned to different knapsacks; but, in this case, split penalties are incurred, according to a Service Level Agreement (SLA) (Sun, Ansari, & Wang, 2016), to take into account either the additional organizational efforts required and/or the level of service lowering (e.g., in terms of response times and delays).

3. Literature review

The *standard* KP has been extensively investigated both from a theoretical and a practical point of view. Readers can refer to Kellerer et al. (2004) and Martello and Toth (1990) for a full-scale presentation of most of the methods and techniques related to the KP family. Furthermore, the many *variants* of the Knapsack Problem has been surveyed in Lin (1998). The MMdKFSP addressed

[1] https://aws.amazon.com/about-aws/global-infrastructure/
[2] https://azure.microsoft.com/en-us/global-infrastructure/availability-zones/
[3] https://aws.amazon.com/ec2/purchasing-options/dedicated-instances/
[4] https://aws.amazon.com/ec2/dedicated-hosts/

in this paper is related to several problems known in the literature; still, to the best of our knowledge, it has not yet been addressed before. Firstly, MMdKFSP can be considered a variant of both the more classic MKP and MdKP. They are two of the most well-known extensions of KP and have received wide attention from the operational research community during the last decades. Surveys on these problems can be found in Chekuri and Khanna (2000), Dell'Amico, Delorme, Iori, and Martello (2019), Ferreira, Martin, and Weismantel (1996), Fréville (2004), and Pisinger (1999). Nevertheless, solving these problems remains an interesting challenge, especially when the size of the instances increases or additional characteristics are considered. Secondly, among the Knapsack variants related to MMdKFSP are those in which the items are *grouped* or *partitioned* into different classes or families and there are consequent adaptations in the objectives and/or in the constraints. These adjustments resulting from applications lead to problems that are the subject of researches and are of considerable importance.

This is the case of the Knapsack Problem with Setups (KPS), which is a generalization of the classical KP, where items are divided into families. Differently from the MMdKFSP case, an individual item can be selected only if a *setup* is payed for the family to which it belongs (Furini, Monaci, & Traversi, 2018; McLay & Jacobson, 2007; Pferschy & Scatamacchia, 2018). Two variants of Knapsack Problems with setups have been presented in Michel, Perrot, and Vanderbeck (2009). Namely, the Multiple-Class Binary Knapsack Problem with Setups (MBKPS) where item weights are assumed to be a multiple of the class weight, and the Continuous Knapsack Problems with Setups (CKS) where each class holds a single item and a *fraction* of an item can be selected while incurring a full setup.

An optimization problem arising in applications, including storage management for multimedia systems, scheduling and logistics is the Class-Constrained Multiple Knapsack (CCMK) (Shachnai & Tamir, 2001), in which sets of items having different sizes and values belong to different *classes*, and each knapsack has a limited capacity. In this case there is an additional constraint on the number of distinct classes of items a knapsack can accommodate. The objective is to maximize the total value of selected items.

In problems inspired by real applications, it emerges that often decision makers are interested in more than one goal, in addition to the more classical KP profit, accounting for fairness, penalties, or even energy consumption associated to the selected items. Often the considered models adopt a *scalarization* of the objectives, keeping the *original* single objective form of the Knapsack Problem. A comprehensive review on the *multi-objective* extension of MdKP and related methods is provided in Lust and Teghem (2012), while Ceselli and Righini (2006) and Della Croce, Pferschy, and Scatamacchia (2019) addressed a Penalized version of Knapsack Problem (PKP) in which each item has associated a constant *penalty* and the objective function is given by the total profit discounted by the largest penalty among the selected items.

Another relevant case is represented by the Multiple-choice Multidimensional Knapsack Problem (McMKP) (Nauss, 1978), which can be informally described as follows. Given a set of items that are divided into several groups and different types of limited resources, each item requires a certain amount of each resource and generates a profit. Contrary to the problem under study, the purpose of the McMKP is to select exactly one item from each group such that the total profit of the selected items is maximized while the consumption of each resource does not exceed the given limit (knapsack constraints). Given its theoretical and practical relevance (Mansi, Alves, Valerio de Carvalho, & Hanafi, 2013; Shojaei, Basten, Geilen, & Davoodi, 2013), the McMKP is receiving increasing attention in recent years and a number of applications, including Quality of Service (QoS) management problem in computer networks and the admission control problem in the adaptive multimedia systems, and effective solution approaches have been proposed in the literature (Chen & Hao, 2014; Han, Leblet, & Simon, 2010; Hifi, Michrafy, & Sbihi, 2004).

Further problems related to MMdKFSP worth mentioning are some variants of MKP for *packing* and *assignment* issues. Namely, those known as Multiple Knapsack Assignment Problem (MKAP), All-or-Nothing Generalized Assignment (AGAP), and the problem of packing groups of items into multiple knapsacks (GMKP). In MKAP, the items to be assigned to the knapsacks are partitioned into disjoint sets as in MMdKFSP but each knapsack may only be assigned items from one of the sets in the partition. The MKAP was introduced in Kataoka (2014), and more recently has been addressed in Martello and Monaci (2018) studying different algorithmic approaches. In Adany et al. (2013) AGAP is introduced pointing out a central application in scheduling advertising campaigns. In this case, items are partitioned into groups, each item has its size and utility, but the latter also depends on the assigned knapsack. The total size of the items in a knapsack, as usual, cannot exceed its capacity, but, in this case, each knapsack can accommodate at most one item from each group. A group of items is satisfied only if all of its items are selected. The goal is to find a feasible packing of a subset of the items in the knapsacks such that the total utility from satisfied groups is maximized. Closer to MMdKFSP appears the problem of packing Groups of Items into Multiple Knapsacks (GMKP) recently addressed in Chen and Zhang (2018). This problem considers a set of items that are partitioned into groups. Each item has an individual size, while the profit, in this case, is associated with groups rather than items. The profit of a group can be earned if and only if all items in the group are selected. They can be placed into different knapsacks, but – unlike MMdKFSP – they are *identical* and, more important, no *split penalties* are considered.

In conclusion of this review, we underline that many problems presented in the literature have some characteristics in common with MMdKFSP, which, nevertheless, features a specific structure and application motivations that justify research activities from a modeling and an algorithmic point of view, and even a proper place in the knapsack family for this problem.

4. Problem statement

The application-based problem studied in this paper can be described as follows. A set $\mathcal{I}$ of $n = |\mathcal{I}|$ items is available for selection, and the generic item is indicated as $i \in \mathcal{I}$ assuming values in $\{1, \dots, n\}$. Items in $\mathcal{I}$ are partitioned in m families $\mathcal{F}_j \subset \mathcal{I}$, with $j \in \{1, \dots, m\}$ (i.e., each item belongs to one and only one family, that is $\mathcal{F}_h \cap \mathcal{F}_k = \emptyset,\ \forall h \neq k$, and $\cup_{j=1}^{m} \mathcal{F}_j = \mathcal{I}$). The set of families is denoted by $\mathcal{F}$.

Each item i represents a computing task or service and is associated with a CPU and a RAM demand, referred to as c_i and r_i, respectively. The global CPU and RAM requirement of a family is equal to the sum of all the demands of the items belonging to it. A profit p_j is associated with each family $\mathcal{F}_j$; such value is deterministic, constant and known in advance. A set of heterogeneous bi-dimensional knapsacks, $\mathcal{K}$, representing the machines on which items are executed, is available. Each knapsack $k \in \mathcal{K}$ is characterized by a capacity value representing the availability of the CPU resource, C_k, and the RAM resource, R_k. The profit p_j is collected only if the corresponding family $\mathcal{F}_j$ is selected, i.e. if all the items belonging to it are assigned to some knapsack. Moreover, items belonging to the same family may be assigned to the same knapsack or to different knapsacks. In the latter case, a penalty denoted by δ_j, which may vary among families, is paid. The goal is to maximize the total *revenue* given by the collected profits curtailed of the incurred family-split penalties. For the sake of clarity, Table 1 reports a complete list of the parameters and variables used in the formulation presented in the next section. Finally,

Table 1
Description of the model parameters and variables.

Parameter	Description
$\mathcal{I}$	Set of items available for selection; $\vert\mathcal{I}\vert = n$
$\mathcal{F}$	Set of item families; $\vert\mathcal{F}\vert = m$
$\mathcal{F}_j$	Set of items belonging to family $j \in \{1, \ldots, m\}$
$\mathcal{K}$	Set of knapsacks; $\vert\mathcal{K}\vert > 1$
p_j	Profit associated with family $\mathcal{F}_j$
δ_j	Split-penalty associated with family $\mathcal{F}_j$
Variable	**Description**
x_j	binary variable assuming value 1 only if family $\mathcal{F}_j$ is selected
y_{ik}	binary variable assuming value 1 only if item i is assigned to knapsack k
z_{jk}	binary variable assuming value 1 only if at least one item belonging to family $\mathcal{F}_j$ is assigned to knapsack k
u_j	binary variable assuming value 0 if all items belonging to family $\mathcal{F}_j$ are assigned to the same knapsack, and 1 otherwise

despite the fact that only two dimensions have been contemplated in the description and formulation of the application-based problem, namely CPU and RAM, it can be straightforwardly extended to cases with a greater number of dimensions. This is shown in an additional experimental campaign to study larger instances and address the scalability issue.

4.1. Problem formulation

The Multiple Multidimensional Knapsack with Family-Split Penalties problem (MMdKFSP) can be formalized through the following Integer Linear Programming formulation:

$$(MMdKFSP) \qquad \max \sum_{j|\mathcal{F}_j\in\mathcal{F}} p_j x_j - \sum_{j|\mathcal{F}_j\in\mathcal{F}} \delta_j u_j \tag{1}$$

Subject to:

$$\sum_{k\in\mathcal{K}} y_{ik} = x_j \quad \forall i \in \mathcal{I}, \quad \forall j | i \in \mathcal{F}_j, \tag{2}$$

$$\sum_{i\in I} c_i y_{ik} \le C_k \quad \forall k \in \mathcal{K}, \tag{3}$$

$$\sum_{i\in I} r_i y_{ik} \le R_k \quad \forall k \in \mathcal{K}, \tag{4}$$

$$z_{jk} \ge \frac{1}{|\mathcal{F}_j|} \sum_{i\in\mathcal{F}_j} y_{ik} \quad \forall j|\mathcal{F}_j \in \mathcal{F}, \quad \forall k \in \mathcal{K}, \tag{5}$$

$$u_j \ge \frac{1}{|\mathcal{K}|-1} \left(\sum_{k\in\mathcal{K}} z_{jk} - 1 \right) \quad \forall j|\mathcal{F}_j \in \mathcal{F}, \tag{6}$$

$$x_j, u_j \in \{0,1\} \quad \forall j|\mathcal{F}_j \in \mathcal{F}, \tag{7}$$

$$z_{jk} \in \{0,1\} \quad \forall j|\mathcal{F}_j \in \mathcal{F}, \quad \forall k \in \mathcal{K}, \tag{8}$$

$$y_{ik} \in \{0,1\} \quad \forall i \in \mathcal{I}, \quad \forall k \in \mathcal{K} \tag{9}$$

The goal of MMdKFSP is to maximize the collected profit reduced by the penalties incurred for splitting families on two or more machines, as stated in (1) keeping, for clarity, the two terms separated. The binary variable x_j assumes value 1 only if family $\mathcal{F}_j$ is selected, whereas the binary variable u_j is set to 0 when all the items of family $\mathcal{F}_j$ are assigned to the same knapsack, and assumes value 1 otherwise (i.e., when a splitting occurs). Constraints (2) ensure that if a family is selected all the items belonging to it are assigned to some knapsack. Note that y_{ik} is a binary variable assuming value 1 only if item i is assigned to knapsack k. Inequalities (3) and (4) imply that capacity limits in terms of CPU and RAM, respectively, are respected for all the knapsacks. The group of Constraints (5) allows to detect whether a knapsack contains items belonging to a given family, while Constraints (6) enable to identify if a family is assigned to a single knapsack, or if it is split on more of them. The binary variable z_{jk} is set to 1 only if at least one item belonging to family $\mathcal{F}_j$ is assigned to knapsack k. Finally, Constraints (7)–(9) specify the variables domain.

5. Combinatorial Benders cuts

The Benders' decomposition (BD) is a mathematical approach, introduced in the pioneering work of Benders (1962), whose core idea is that fixing a set of variables, those that supposedly make a Mixed Integer Programming model (MIP) hard to solve, may strongly simplify the problem. This approach decomposes the problem at issue into a Master Problem (MP), in which only a subset of the overall set of decision variables is considered, and a Slave Problem (SP), containing the remaining variables, and solves them iteratively in succession. The MP is solved first, then the SP is generated by temporarily fixing the variables values of the MP, and solved for the remaining variables. Finally, based on the outcome of the SP, one or more cuts are generated and added to the MP, preventing the MP from exploring specific areas of the search space. In the classical application of BD to MIPs, the SP is a Linear Programming problem, (LP), and its dual solution is exploited to derive cuts to be added to the MP. Successively BD has been generalized and extended to problems in which the SP is not required to be a LP (Geoffrion, 1972). More than 30 years later, the concept of Logic Based Benders decomposition has been introduced by Hooker and Ottoson (2003). In this approach, in the MP are considered only those variables that contribute directly to the objective function, while the others are delegated to the SP. In this way, the SP becomes a pure feasibility problem; whenever the SP turns to be infeasible one or more cuts are generated to cut off detected infeasible solutions from the search space of the MP. Once a feasible SP is detected, than the overall solution obtained is proved to be optimal.

In Codato and Fischetti (2006) a particular case of Logic Based Benders decomposition, named Combinatorial Benders decomposition, has been presented. This approach is specifically suited for MIP models, involving binary variables and a large number of logical implications through Big-M constraints. In this scenario, whenever to a particular combination of the MP variables corresponds an infeasible SP, a Combinatorial Benders cut (CBC) is added to force to 0 at least one of the variables assuming value 1 in the MP' solution. Stronger cuts can be obtained identifying the subset of variables responsible for the infeasibility through the detection of a Minimum Infeasible Set (MIS). Such cuts are stronger because they tend to cut off from the MP search space several solutions at a time, sensibly speeding-up the convergence toward an optimal solution. Nonetheless, it is worth noting that the convergence of the algorithm is always guaranteed even adopting only standard CBC cutting off one solution at a time (Chu & Xia, 2004; Codato & Fischetti, 2006).

Several successful applications of CBC to real problems arising in different contexts can be found in the literature. In Bai and Rubin (2009), a toll facilities location problem is addressed. While in Côté, Dell'Amico, and Iori (2014) a Strip Packing problem arising in many application field as container loading operations, cutting operations and scheduling is effectively solved by CBC. Applications to quayside operations at container terminals are reported in Cao, Lee, Chen, and Shi (2010) and Chen, Lee, and Cao (2012), whereas in Verstichel, Kinable, De Causmaecker, and Vanden Berghe (2015) another problem arising in port logistics, the lock scheduling, is addressed. Other innovative applications regards beam intensity modulation in radiotherapy (Taşkin & Cevik, 2013) and assembly line balancing (Akpinar, Elmi, & Bektaş, 2017).

All the above cited problems share a common structure where the variables can be partitioned into two subsets: the first one containing variables contributing directly to the objective function (addressed in the MP), and the second containing variables only acting on the feasibility of the solution (addressed in the SP). Interestingly, one of the innovative aspects of the approach presented in this work is that, differently from these cases, the addressed problem features variables that can be partitioned into two groups, both contributing to the objective function. Variables in the first group are involved into profit collection maximization (first term in (1)), while variables in the second group are responsible for the penalties (second term of (1)). In this way, the objective function of the MP only concerns the collected profit maximization overlooking the incurred penalties minimization. This entails that the objective function value obtained solving the MP is an Upper Bound (UB) of the actual value corresponding to the MP variables optimal configuration. The SP role, instead, is twofold: firstly, it is used to detect infeasibility as in the classical CBC, and secondly, to allow computing the actual value of the objective function. Consequently, two classes of cuts can be generated at each iteration: the classical feasibility cuts (F-CUTS), and optimality or penalty cuts (P-CUTS).

The first kind of cuts is obtained by the CBC described in this section, while the second one is used to impose that a penalty must be added to the MP objective function to detect the actual corresponding objective function value of the original problem. More details about this newly introduced approach are reported in the following subsection.

5.1. A combinatorial Benders cuts method for the MMdKFSP

The aim of this section is to introduce a particular Benders decomposition for the MMdKFSP. In a nutshell, an iterative algorithm is proposed that at each iteration q generates a master problem MP_q possibly adding suitable and specific cuts obtained from the solution of SP_{q-1}. For the sake of clarity, Algorithm 1 reports in pseudo-code the CBC method described as follows. In a first phase, indicated in Algorithm 1 as *Optim_Phase*, the method sequentially solves the current master and slave problems.

At the first iteration (i.e., $q = 1$), the master problem MP_1 consists of a relaxation of the ILP formulation in which a single multi dimensional meta-knapsack is considered having, for each dimension, a capacity equal to the sum of the capacities of all the knapsacks included in the original problem. Consequently, two parameters $\tilde{C} = \sum_{k\in\mathcal{K}} C_k$ and $\tilde{R} = \sum_{k\in\mathcal{K}} R_k$ are defined. Moreover, since a single meta-knapsack is considered, the splitting process loses its meaning and, as a consequence, the problem can be further simplified by consolidating the items of each family into a single indivisible meta-item. Thus, each family $\mathcal{F}_j \in \mathcal{F}$ is replaced by a new meta-item with CPU and RAM requirements equal to the following:

$$\hat{c}_j = \sum_{i\in\mathcal{F}_j} c_i;\ \text{and}\ \hat{r}_j = \sum_{i\in\mathcal{F}_j} r_i \tag{10}$$

Algorithm 1 Combinatorial Benders Cuts for MMdKFSP (CBC).

```
MP_1 = getRelaxedFormulation(MMdKFSP)
INITIALIZE $q = 1, \rho_1 = 0, A_1 = 0, Optim\_Phase = TRUE, Solution = \emptyset$
while Optim_Phase do
    $\bar{x}^q$ = solve($\text{MP}_q$)
    $\Omega^q_{MP}$ = optValue($\text{MP}_q$, $\bar{x}^q$)
    $\text{SP}_q$ = MMdKFSP $\wedge\ x_j = \bar{x}^q_j \quad \forall j|\mathcal{F}_j \in \mathcal{F}$
    $\bar{s}^q$ = solve($\text{SP}_q$)
    $\Omega^q_{SP}$ = optValue($\text{SP}_q$, $\bar{s}^q$)
    $a_{q+1} = \sum_{j|\mathcal{F}_j\in\mathcal{F}} \bar{x}^q_j$
    if isFeasible($\bar{s}^q$) then
        if $\Omega^q_{SP} = \Omega^q_{MP}$ then
            $Solution = \bar{s}^q$
            $Optim\_Phase = FALSE$
        else
            $\rho_{q+1} = \sum_{j|\bar{x}^q_j=1} \delta_j u_j$
            $\text{MP}_{q+1}$ = addPenalityTerm($\text{MP}_q$, $\rho_{q+1}$, $a_{q+1}$)
        end if
    else
        $\rho_{q+1} = 0$
        $\text{MP}_{q+1}$ = addCut($\text{MP}_q$, $\rho_{q+1}$, $a_{q+1}$)
    end if
    $q = q + 1$
end while
```

Accordingly, the resulting MP_1 can be described as a classical MdKP, which is much simpler to solve than MMdKFSP as, among other things, it involves a sensibly smaller number of decision variables. The formulation of the MP_1 is described in the following.

$$(MP_1)\ \max \sum_{j|\mathcal{F}_j\in\mathcal{F}} p_j x_j \tag{11}$$

Subject to:

$$\sum_{j|\mathcal{F}_j\in\mathcal{F}} \hat{c}_j x_j \le \tilde{C} \tag{12}$$

$$\sum_{j|\mathcal{F}_j\in\mathcal{F}} \hat{r}_i x_j \le \tilde{R} \tag{13}$$

$$x_j \in \{0, 1\} \quad \forall j|\mathcal{F}_j \in \mathcal{F} \tag{14}$$

The objective to maximize is the collected profit as stated in (11). Constraints (12) and (13) imply that the capacity limit of the consolidated meta-knapsack, in terms of CPU and RAM, is respected, while the variables domain is specified by (14).

Once an optimal solution $\bar{x}^1 = \{\bar{x}^1_1, \bar{x}^1_2, \ldots, \bar{x}^1_m\}$ for the MP_1 has been identified, the collection of selected families has to be tested for feasibility for the original MMdKFSP and, if it passes, then it is necessary to check if any penalty is incurred. To this end, a Slave Problem (SP_1) is defined that solves the MMdKFSP fixing the values of the x_j variables to those provided by the optimal solution of the MP_1; the resulting problem can therefore be written imposing into the formulation provided in Section 4.1 the following constraints:

$$x_j = \bar{x}^1_j \quad \forall j|\mathcal{F}_j \in \mathcal{F} \tag{15}$$

Clearly, if the solution returned by SP_1, referred to as $\bar{s}^1$, is feasible and the corresponding value of the objective function for SP_1, say Ω^1_{SP}, is equal to that of MP_1, say Ω^1_{MP}, then $\bar{s}^1$ is optimal for the MMdKFSP problem as well. Diversely, if $\Omega^1_{SP} < \Omega^1_{MP}$, $\bar{s}^1$ is clearly sub-optimal as one or more families have been split incurring in the following penalty:

$$\rho_2 = \sum_{j|\bar{x}^1_j=1} \delta_j \tag{16}$$

which is equal to sum of the penalties associated to the MP_1 families selection. To take into account the discrepancy between the expected and the actual value of the objective function (i.e., Ω^1_{MP} Vs Ω^1_{SP}), a MP_2 is created adding the penalty term (16) in the objective function (11) that is activated only if the corresponding set of families is selected also by the MP_2 (i.e., $\bar{x}^2 = \bar{x}^1$). In Algorithm 1, the function *addPenaltyTerm* performs these actions. The desired effect is obtained adding into MP_2 also the following constraint that activates the binary variable ν_2 if all the families, chosen by the MP_1, are selected also by the MP_2:

$$\sum_{j|\bar{x}^1_j=1} x_j \le a_2 - 1 + \nu_2 \tag{17}$$

where $a_2 = \sum_{j|\mathcal{F}_j\in\mathcal{F}} \bar{x}^1_j$. The objective function of the MP_2 becomes:

$$\max \sum_{j|\mathcal{F}_j\in\mathcal{F}} p_j x_j - \rho_2 \nu_2 \tag{18}$$

If the SP_1 is infeasible, then the following cut is introduced (function *addCut* in Algorithm 1) to refrain the MP_2 form generating the same infeasible solution:

$$\sum_{j|\bar{x}^1_j=1} x_j \le a_2 - 1 \tag{19}$$

This procedure is repeated until, for a certain iteration t, it results $\Omega^t_{SP} = \Omega^t_{MP}$, and therefore $\bar{s}^t$ is optimal for the MMdkFSP, and therefore *Optim_Phase* is no longer executed in Algorithm 1. For each iteration $q > 1$ the MP_q can be formulated as follows:

$$(MP_q) \max \sum_{j|\mathcal{F}_j\in\mathcal{F}} p_j x_j - \sum_{h=2}^{q} \rho_h \nu_h \tag{20}$$

Subject to:

$$\sum_{j|\mathcal{F}_j\in\mathcal{F}} \hat{c}_j x_j \le \tilde{C} \tag{21}$$

$$\sum_{j|\mathcal{F}_j\in\mathcal{F}} \hat{r}_i x_j \le \tilde{R} \tag{22}$$

$$\sum_{j|\bar{x}^{h-1}_j=1} x_j \le a_h - 1 + \nu_h \quad \forall h \in \{2,\dots,q\} |\ \bar{s}^{h-1}\ feasible \tag{23}$$

$$\sum_{j|\bar{x}^{h-1}_j=1} x_j \le a_h - 1 \quad \forall h \in \{2,\dots,q\} |\ \bar{s}^{h-1}\ infeasible \tag{24}$$

$$x_j \in \{0,1\} \quad \forall j|\mathcal{F}_j \in \mathcal{F} \tag{25}$$

$$\nu_h \in \{0,1\} \quad \forall h \in \{1,\dots,q\} \tag{26}$$

$$\nu_1 = 0 \tag{27}$$

The convergence of the overall algorithm towards an optimal solution is guaranteed in a finite number of steps as in the more general cases (Chu & Xia, 2004; Codato & Fischetti, 2006).

5.2. Pre-processing procedure

This section aims at illustrating a pre-processing heuristic method developed to possibly identify in advance families to be necessarily split in any feasible solution. This method can be applied on both the ILP model and the CBC alike, and it works as follows. Firstly, the procedure identifies the knapsacks with the highest capacity in terms of CPU, C_{max} and RAM, R_{max}. Then, for each family $\mathcal{F}_j$ the method checks if the related CPU and RAM demands, that is $\hat{c}_j$ and $\hat{r}_j$, are lower than C_{max} and R_{max}, respectively. If this test fails, it means that no knapsack comes with enough capacity to contain the whole family $\mathcal{F}_j$. Hence, if $\mathcal{F}_j$ were selected, it would be necessarily split among two or more knapsacks. This implies that the actual profit collected selecting $\mathcal{F}_j$ would be $p_j - \delta_j$ since splitting penalty δ_j would occurs.

In order to exploit this information, for all the families $\mathcal{F}_j$ necessarily needing to be split, we *artificially* re-define $p_j = p_j - \delta_j$, and then we impose $\delta_j = 0$, since the penalty is already embedded in the profit evaluation. The effect of this procedure on the CBC is that, at each iteration the solution of the MP becomes a tighter bound for the original problem, strongly reducing the number of P-CUTS to be added. In fact, without the pre-processing, a penalty cut occurs every time a family is split, while using the pre-processing a P-CUT occurs only if the split family was not identified by the pre-processing. The effect of the procedure on the ILP model is to implicitly remove (i.e., impose to 0) from the set of decision variables, the split variables, u_j, for all the families identified by the pre-processing. In fact, being the split penalty $\delta_j = 0$ for such families, the corresponding u_j does not have any effect on the objective function. A more formal description of this procedure is reported in Algorithm 2.

Algorithm 2 Pre-processing Heuristic.

```
Define $C_{max} = \max_{k\in\mathcal{K}} C_k$; and $R_{max} = \max_{k\in\mathcal{K}} R_k$
for each family $\mathcal{F}_j \in \mathcal{F}$ do
    if $\hat{c}_j \ge C_{max}$ or $\hat{r}_j \ge R_{max}$ then
        set $p_j = p_j - \delta_j$
        set $\delta_j = 0$
    end if
end for
```

The complexity of this procedure can be quantified in $O(|\mathcal{K}| + |\mathcal{F}|)$ since $|\mathcal{K}|$ operations are needed to identify C_{max} and R_{max}, while $2 * |\mathcal{F}|$ comparison are needed, in the worse case, to determine if a family must necessarily be split in any feasible solution. Both the procedure and its complexity analysis can be directly extended to higher dimensional cases.

6. Experimental analysis

This section illustrates the computational experiments designed and conducted to assess the effectiveness of the proposed algorithm, analyze its behavior, and compare the results obtained with respect to those attainable by the baseline ILP formulation solved by a state-of-the-art commercial solver. All the experiments reported in this paper have been carried out on a Intel-i7-5500U processor (2.4 gigahertz clock speed) operating under Windows 7, and equipped with 16 gigabytes of RAM. The code has been developed in Xpress-mosel language and run on a single thread. In the CBC approach, both the MP and the SP are solved by the same ILP solver Xpress 7.9. Results are compared with those obtained by Xpress 7.9 used to solve the MMdKFSP ILP formulation introduced in Section 4.1. All reported CPU times for the computations are expressed in seconds. In the following Section 6.1 the experimental setting is described in terms of benchmark instance sets and scenarios considered in the computational campaign. Results are reported, analyzed and discussed in Section 6.2, which also contains a comparison with the commercial solver considering also an additional experimental campaign to study larger and multidimensional instances and address the scalability issue.

6.1. Experimental setup

In order to evaluate the appropriateness of the approach described in the previous sections, a number of instances have been

Table 2
Set of test instances A–B–C.

SET	$\|\mathcal{K}\|$	$\|\mathcal{I}\|$	$\|\mathcal{F}\|$
A	3	400	6÷9
B	3	500	8÷12
C	3	600	9÷14

Table 3
Set of test instances D–E–F.

SET	$\|\mathcal{K}\|$	$\|\mathcal{I}\|$	$\|\mathcal{F}\|$
D	5	400	7÷9
E	5	500	8÷10
F	5	600	10÷12

Table 4
Set of test instances G–H–I.

SET	$\|\mathcal{K}\|$	$\|\mathcal{I}\|$	$\|\mathcal{F}\|$
G	10	400	6÷8
H	10	500	8÷11
I	10	600	10÷13

generated with the aim of covering different scenarios of interest in distributed computing applications on a departmental or local scale. It has been necessary to do this as the instances available in the literature are difficult to adapt and extend to our model. The first direct consequence of this choice is to be able to propose a reference to other researchers for future comparison.

The main characteristics of the benchmark are summarized in Tables 2–4, in which the first column defines a subset of 10 instances characterized by the same number of knapsacks {3, 5, 10} (column two) and items {400, 500, 600} (column three). Both $|\mathcal{F}|$ and $|\mathcal{F}_j|$ are iteratively determined in the instance generation procedure and the range of the number of families $|\mathcal{F}|$ (the actual value depending on the number of items in each family) is reported in column four. Specifically, all the items are generated in a single pass and then iteratively assigned to families. Being ψ_h the number of items assigned at iteration h, the assignment at iteration $h+1$ is achieved by drawing from an uniform distribution in the range $[\min(25, |\mathcal{I}| - \psi_h),\ \min(100, |\mathcal{I}| - \psi_h)]$, considering $\psi_0 = 0$. Each row in Tables 2–4 represents a set of ten different instances which are described in more details in the Electronic Companion of this paper containing some supplementary material.

Regarding the items' requirements, 2 dimensions have been considered in this benchmark (corresponding to CPU and RAM) both scaled to be an integer value uniformly distributed in the range [1, 100]. Similarly, for each family $\mathcal{F}_j \in \mathcal{F}$ a profit p_j is generated for a uniform distribution in the range [1, 100], while the penalty value δ_j depends on the profit by picking randomly from a uniform distribution in the range $[0.2p_j,\ 0.5p_j]$.

Finally, the capacity of each knapsack is obtained by applying the following formulae:

$$C_k = \frac{\alpha_k}{|\mathcal{K}|}\sum_{i\in\mathcal{I}} c_i; \text{ and } R_k = \frac{\beta_k}{|\mathcal{K}|}\sum_{i\in\mathcal{I}} r_i; \qquad \forall k \in \mathcal{K} \tag{28}$$

The way the values of α_k and β_k are selected defines the different scenarios explained in the following.

The total number of instances in this baseline set amounts to 90. This baseline has been used to generate 4 different scenarios (from SC-1 to SC-4) characterized by different knapsacks capacity settings, that is values of α_k and β_k. Consequently, the total number of instances considered in the experimental campaign reported in this study is 360.

In the first scenario (SC-1) the knapsacks' capacities have homogeneous, yet different, values obtained extracting α_k and β_k from $\overline{U}(0.45, 0.55), \forall k \in \mathcal{K}$. In other words, the total knapsack capacity can contains roughly half of the items. This scenario aims to investigate a situation in which the family selection is challenging. In fact, homogeneous capacities generally make the choice of families more difficult, while the presence of very high or very low values of capacities may induce forced choices of families, making an instance simpler. All the successive scenarios adopt a basic unbalanced pattern with a knapsack larger than the others having capacity equal to the maximum compound demand per dimension calculated over all families (i.e., $\hat{c}_j$ and $\hat{r}_j$), that is: $C_1 = \max_{j|\mathcal{F}_j\in\mathcal{F}} \hat{c}_j$, and $R_1 = \max_{j|\mathcal{F}_j\in\mathcal{F}} \hat{r}_j$.

Table 5
Instance parameters.

Parameter	Values
$\|K\|$	{3, 5, 10}
$\|I\|$	{400, 500, 600}
$\|\mathcal{F}\|$ and $\|\mathcal{F}_j\|$	Iteratively determined
r_i, c_i, p_j	$\overline{U}(1, 100)$
δ_j	$\overline{U}(0.2p_j, 0.5p_j)$
α_k, β_k	SC-1: $\overline{U}(0.45, 0.55)\ \forall k \in \mathcal{K}$,
	SC-2: $\overline{U}(0.27, 0.33)\forall k \in \mathcal{K} - \{1\}$
	SC-3: $\overline{U}(0.09, 0.11)\ \forall k \in \mathcal{K} - \{1\}$
	SC-4: $\overline{U}(0.45, 0.55)\ \forall k \in \mathcal{K} - \{1\}$

Furthermore, in these cases the other knapsacks' capacity is reduced to make the assignment more challenging with respect to SC-1. More specifically, they have homogeneous values obtained extracting α_k and β_k from $\overline{U}(0.27, 0.33), \forall k \in \mathcal{K} - \{1\}$ for SC-2; $\alpha_k, \beta_k \in \overline{U}(0.09, 0.11),\ \ \forall k \in K - \{1\}$ for SC-3, and $\alpha_k, \beta_k \in \overline{U}(0.45, 0.55),\ \ \forall k \in K - \{1\}$, for SC-4 respectively. As a result, in SC-4 the first knapsack is about twice as capacious as the others. This scenario was created to obtain instances able to contrast the impact of the pre-processing phase, and therefore they tend to be more difficult for the algorithm, in particular with respect to SC-1, of which retain the other characteristics. In fact, it is expected that, according to the pre-processing procedure, few (or even none) families necessarily need to be split (i.e., they can be contained in the largest knapsack) whereas the feasible solutions (possibly) contain many split families. Table 5 provides an overview of the parameters with their values.

6.2. *Results*

The algorithmic approach based on Benders cuts proposed in Section 5.1 has been tested on the test-sets and scenarios described in Section 6.1. In particular, the CBC algorithm and the ILP model – both equipped with the pre-processing heuristic method illustrated in Algorithm 2 – have been run on each of the 360 instances allowing a maximum computation time of 3600 seconds. Detailed results obtained for all the single instances considered in the experiments are reported in the Electronic Companion. An aggregate descriptive analysis of the results achieved in the experiments is, instead, summarized in Tables 6–9 referring to the four scenarios introduced in Section 6.1. In the tables are reported the averaged results for each set of instances indicated in the first column (*SET*). Further, each table contains two sub-tables reporting the results obtained in the considered scenario by the ILP formulation (*MODEL*) and the proposed algorithm (*CBC*), respectively. The sub-table devoted to the solver results contains two columns. The first (*#OPT*) reports, for each set, the number of instances solved to the optimality in the allowed computation time, whereas the second column (denoted by *TIME*) shows the required processing time expressed in seconds. The second sub-table (*CBC*) is dedicated to the aggregated results achieved by the CBC algorithm. In this case, the group contains four columns. The first two columns show information similar to those just introduced for the MODEL sub-

Table 6
Aggregated results for scenario SC-1.

	MODEL		CBC			
SET	#OPT	TIME(seconds)	#OPT	TIME(seconds)	#F-CUTS	#P-CUTS
A	10	0.54	10	0.3	0 (0)	2 (3)
B	10	2.02	10	0.57	0 (0)	5 (6)
C	10	4.97	10	1.8	0 (0)	6 (9)
D	10	0.44	10	0.11	0 (0)	0 (0)
E	10	0.74	10	0.11	0 (0)	0 (0)
F	10	1.62	10	0.89	0 (0)	1 (2)
G	10	0.59	10	0.06	0 (0)	0 (0)
H	10	0.56	10	0.18	0 (0)	1 (1)
I	10	4.18	10	2.83	0 (0)	0 (0)
Avg.	10	1.74	10	0.76	0 (0)	1.67 (2.33)

Table 7
Aggregated results for scenario SC-2.

	MODEL		CBC			
SET	#OPT	TIME(seconds)	#OPT	TIME(seconds)	#F-CUTS	#P-CUTS
A	10	0.96	10	0.27	0 (0)	4 (13)
B	10	1.81	10	0.83	0 (0)	9 (20)
C	10	2.85	10	1.42	0 (0)	9 (16)
D	10	6.03	10	5.18	0 (0)	10 (47)
E	10	8.07	10	4.2	0 (0)	10 (55)
F	10	24.83	10	19.69	0 (0)	10 (80)
G	10	21.26	10	15.75	0 (0)	10 (59)
H	10	219.15	10	88.3	2 (3)	10 (163)
I	10	284.28	10	199.27	1 (1)	10 (255)
Avg.	10	62.36	10	37.21	0.33 (0.44)	9.11 (73.89)

Table 8
Aggregated results for scenario SC-3.

	MODEL		CBC			
SET	#OPT	TIME(seconds)	#OPT	TIME(seconds)	#F-CUTS	#P-CUTS
A	10	0.6	10	0.03	1 (1)	8 (10)
B	10	0.73	10	0.12	0 (0)	9 (18)
C	10	1.60	10	0.32	0 (0)	10 (17)
D	10	1.29	10	0.13	0 (0)	9 (12)
E	10	2.6	10	0.24	0 (0)	10 (18)
F	10	4.65	10	1.08	0 (0)	10 (43)
G	10	4.55	10	1.35	0 (0)	9 (18)
H	9	370.92	10	3.29	0 (0)	9 (33)
I	10	13.76	10	5.38	1 (1)	10 (36)
Avg.	9.89	44.52	10	1.33	0.22 (0.22)	9.33 (22.78)

Table 9
Aggregated results for scenario SC-4.

	MODEL		CBC			
SET	#OPT	TIME(seconds)	#OPT	TIME(seconds)	#F-CUTS	#P-CUTS
A	10	1.24	10	0.8	0 (0)	4 (10)
B	10	2.74	10	1.42	0 (0)	6 (7)
C	10	4.68	10	1.97	0 (0)	6 (7)
D	10	4.5	10	6.75	0 (0)	10 (46)
E	10	10.53	10	19.18	0 (0)	10 (59)
F	10	69.54	10	66.46	0 (0)	7 (20)
G	10	96.8	10	198.99	0 (0)	9 (110)
H	9	659.92	9	950.6	1 (1)	10 (183)
I	6	1948.78	10	1534.82	0 (0)	10 (374)
Avg.	9.44	310.97	9.89	308.99	0.11 (0.11)	8 (85.44)

table, that is relating to the number of instances resolved to the optimum and to the related execution time. The last two columns (*#F-CUTS* and *#P-CUTS*, respectively) contain further interesting information about the behavior of the CBC algorithm giving two measures of the F-CUTS and P-CUTS utilization solving each set of instances. Each element of these columns indicates, for the considered set, in how many instances cuts have been generated and, in brackets, the total number of cuts generated for the set. Finally, the last row in Tables 6–9 contains for each column the averages over the considered sets.

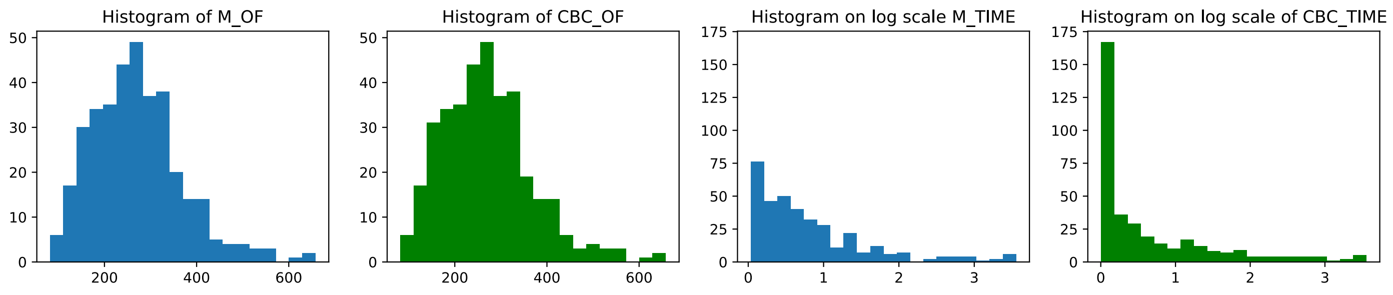

Fig. 1. Histograms of OF and TIME (in log scale) variables for both MODEL and CBC.

6.2.1. *Discussion*

In this section, the results of the experimental campaign carried out to assess the soundness of the proposed approach are analyzed and discussed.

To this end, an accurate statistical analysis has been carried out to evaluate the performance of the two approaches, comparing them both from the viewpoint of the quality of the returned solution as well as of the required computation times. It is worth emphasizing at this point that the analysis carried out is indeed comparative and limited to the two approaches in issue, yet one of them represents a top level out-of-the-box optimization solutions and, therefore, it offers a solid reference point for comparison, giving a general value to the results of the analysis.

At the outset, in order to evaluate the validity of our approach, two independent variables must be analyzed: the value of the objective function (indicated as *OF*) and the calculation time (*TIME*). As far as the value of the objective function is concerned, to begin with a graphic analysis has been carried out plotting the results of the two solutions. It became evident in this way that the results are extremely similar and that neither sample follows a normal distribution (see Fig. 1). To confirm this, we have performed a Kolmogorov-Smirnov test (Sprent, Smeeton, Chatfield, & Zidek, 2000), which rejected the null hypothesis that the samples have been selected from normal populations and consequently imposed the use of a non-parametric test (on related samples) to identify any statistically significant differences. So the Wilcoxon signed-rank test (Sprent et al., 2000) with zero split rank has been used, which substantially has failed to reject the null hypothesis that two related paired samples come from the same distribution. Ultimately, such an outcome was to be expected since the two competing approaches were given a fairly long time to solve the problem. In whichever case, at macroscopic level, it can be observed that CBC has behaved marginally better succeeding in delivering better results in 7 cases corresponding to 1.94% of the total, whereas MODEL has resulted preferable in only one out of 360 (accounting for 0.28%). Similarly, it has been possible to reject the normality hypothesis for the independent variable TIME of both MODEL and CBC alike (as evident form Fig. 1). Similar graphical analyses for each single group of instances are reported in the Electronic Companion, which also reports the Kernel Density Estimations of the TIME variable of MODEL and CBC. The diagram suggests lower computation times for CBC and a higher number of cases where MODEL has exceeded the time limit of 3600 seconds. The Wilcoxon signed-rank test confirmed that exists a statistically significant difference between the approach (in favor of CBC) as far as the execution time is concerned. The statistical tests have been carried out with a confidence level of 0.01; due to the fact that 4 tests have been carried out on the same data set, it is necessary to perform the Bonferroni correction that brings the real confidence level to 0.05.

Proceeding by analyzing the different scenarios individually, we can observe in Table 6 that in the first scenario (SC-1) both MODEL and CBC are able to solve all the instances within the allowed time of 3600 seconds. As expected, these instances, being less constrained in terms of knapsack capacity, are in a sense *simpler* to solve. Nevertheless, the CBC algorithm offers a better performance in terms of computational time (requiring on average 0.76 seconds) compared to the solver (which needs on average 1.74 seconds) which takes more than twice as long. The statistical relevance of these results is also supported by an analysis based on the Wilcoxon test which has been singularly applied to all the scenarios. Looking at the results in detail (available in the Electronic Companion), it is noted that only in 6.7% of the instances the solver is faster than the CBC algorithm. Being both algorithms equipped with the same pre-processing procedure, the comparison is fair, and its outcome clearly due to the behavior of the two different optimization approaches. They spend more computational effort for the sets containing larger instances, i.e., those with more knapsacks and more items. In cases with the same number of knapsacks there is a noticeable increase in the calculation time passing to the sets with a greater number of items. Regarding the behavior of the CBC algorithm, it can be noted that F-CUTS were not generated at all in SC-1, while in 16.7% of instances were generated (one or more) P-CUTS. It emerges that in this scenario, with weakly constrained and homogeneous sized knapsacks, both the impact of the pre-processing and the onset of cuts remain rather limited.

Passing to the results obtained in the second scenario (SC-2) we note (in Table 7, and in the Electronic Companion) that again both the algorithms solve all the instances within the allowed computation time. Nevertheless, these instances are more constrained in terms of knapsack capacity, and are slightly more *difficult* to solve due also to the unbalanced pattern for the capacities. In fact, the computation time increases with respect to SC-1 for both the algorithms. The CBC algorithm marks again the better performance in terms of execution time (37.21 seconds on average) whereas the solver, on average, runs in 63.25 seconds. Thus, in SC-2 the difference in behavior, although sensitive and statistically relevant, appears slightly reduced with respect to that observed in SC-1, and in 80% of the instances CBC algorithm is faster than the solver. It is confirmed that the algorithms require longer times for the sets with the larger instances. In this scenario, CBC algorithm generates one or more F-CUTS in 3.33% of instances, whereas in 91.11% of instances were generated P-CUTS. In SC-2 we can notice a greater generation of P-CUTS with respect to SC-1 with the number of cuts that grows with the size of the instances. Furthermore, in this case we can observe that in only 8.9% of the instances no cuts were generated, and the average was 7.89 cuts per instance. In SC-2, having more constrained instances, the effect of the application of cuts by CBC algorithm is more pronounced with respect to the previous scenario SC-1. This aspect is more accentuated in the third scenario (SC-3) that contains more constrained instances which are – in general – *harder* to solve. In fact, analyzing the results obtained in SC-3, we observe (in Table 8, and in the Electronic Companion) that both the algorithms are called to make

Table 10
Aggregated results for high dimensional instance sets $M1$–$M4$.

	MODEL		CBC			
SET	#OPT	TIME(seconds)	#OPT	TIME(seconds)	#F-CUTS	#P-CUTS
$M1$ ($l = 2$)	10	648.39	10	207.48	2 (3)	10 (240)
$M2$ ($l = 4$)	10	910.51	10	320.27	7 (14)	10 (304)
$M3$ ($l = 6$)	8	1075.93	10	321.07	7 (43)	10 (278)
$M4$ ($l = 8$)	8	1001.15	10	424.63	8 (68)	10 (346)
Avg.	9	908.995	10	318.362	6 (32)	10 (292)

a greater effort. However, this leads to very different outcomes in the two cases. Firstly, there is a noticeable increase in the difference in computation time: the solver takes 44.52 seconds on average, whereas CBC only 1.33 seconds, and only in one instance the solver was faster than CBC. This result is due, on one hand, to the classical branch and bound procedure of the solver which sometimes has difficulty to *certify* the optimality of the solutions found and therefore to *close* the instances; and, on the other hand, to the advantages offered by the generation of cuts by CBC algorithm. They are used in 93.3% of the instances for the P-CUTS type (the average is 2.28 cuts per instance) and in 2.22% for the F-CUTS. Secondly, we note that for one instance (namely, in the set H) the solver is not able to complete the optimization in the 3600 seconds allowed, while CBC algorithm successfully complete its work. In the fourth scenario SC-4, results (reported in Table 9, and in the Electronic Companion) show how both algorithms encounter greater difficulties due to the lesser effect of the pre-processing procedure. In fact the latter, considering the presence of large knapsacks, tends to identify a few families to split. The computation times increase and are similar for the two algorithms (i.e., 310.97 seconds for the solver, and 309.00 seconds for CBC). However, CBC algorithm solves exactly 98.89% of the instances while the solver only reaches 94.44%. The computational burden for CBC is highlighted by the generation of a high number of P-CUTS (more than 9 per instance, on average) mainly due to the reduced effectiveness of the pre-processing procedure in this scenario. The very small number of F-CUTS incurred can be explained by the fact that the size of the knapsacks is very big with respect to the size of the items and this generates a granularity effect which makes easy, once the families have been selected, to find a feasible assignment of items to the knapsacks. To support this hypothesis, we can observe that in scenario SC-3, where knapsacks size is sensibly smaller, the occurrence of F-CUTS increases.

Overall it appears that CBC algorithm is able to take a much better advantage of the support offered by the pre-process phase, and even when this is ineffective, is able to achieve better performances with respect to the baseline ILP formulation. Moreover, one thing not to overlook is that the computational efforts are related to the total available capacity but also, and in a sensitive way, to how this capacity is allocated among the different knapsacks.

6.2.2. *High dimensional instances and scalability analysis*

In order to study the behavior of the proposed algorithmic approach on high dimensional instances and investigate its scalability, a different set of instances has been generated. Such instances are characterized by the number of knapsacks $|\mathcal{K}| = 10$ and the number of items $|\mathcal{I}| = 600$. Every knapsack and item feature an equal number of dimensions l (constant within the same instance) that assumes value in the set $\mathcal{D} = \{2, 4, 6, 8\}$. A detailed description of these new instances is included in the Electronic Companion in which 4 sets $M1$–$M4$ of 10 instances each are introduced for a total of 40 cases. The Electronic Companion also contains a description of some slight adaptations of the proposed approach necessary to extend the number of dimensions beyond 2.

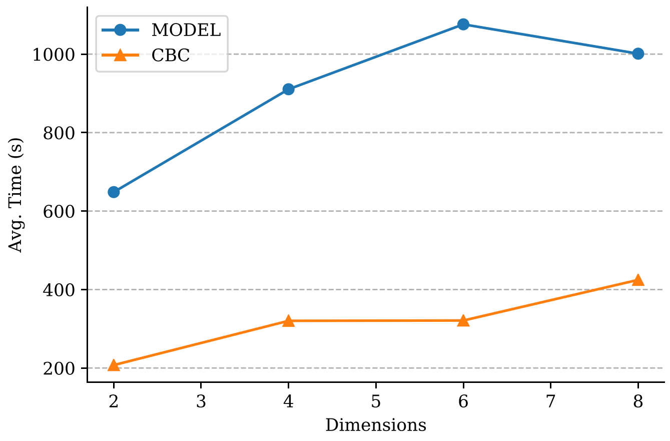

Fig. 2. High dimensional instances: computation times for MODEL and CBC.

Table 10 reports aggregated results obtained for all the 40 considered high dimensional instances of MMdKFSP organized according to the different $M1$–$M4$ groups characterized by different dimensions l. The composition of Table 10 is very similar to that adopted to show aggregated results in the previous campaign of experiments. The indication of the group of instances appear in the first part of the table reporting the set and, in brackets, the number of dimensions l. The better behavior of CBC, summarized also in Fig. 2, is more pronounced with respect to computation times in the case of large-size instances. The results are supported by statistical tests and show the better behavior of CBC both in terms of effectiveness and efficiency. In these experiments, CBC algorithm generates more F-CUTS and their number grows with dimension l, whereas in all instances have been generated several P-CUTS. Detailed results for each of the 40 instances are included in the Electronic Companion where further graphical analyses of the results are also offered.

To consider the impact of the number of knapsacks and items on the behavior of the algorithms we conducted two further tests. Firstly, we consider instances characterized by the number of knapsacks $|\mathcal{K}| = \{10, 15, 20\}$, and a fixed number of items $|\mathcal{I}| = 600$. Then, a second test is conducted on instances with a fixed number of knapsacks $|\mathcal{K}| = 10$, and different number of items $|\mathcal{I}| = \{600, 800, 1000\}$. In both cases, every knapsack has a number of dimensions $l = 2$, and 3 groups of 10 instances have been considered, assuming the set $M1$ (and the related results) as reference. Tables 11 and 12, using the same structure adopted in the previous tables, report aggregated (i.e., each row refers to a group of 10 instances) results obtained for these two computational tests. The results show that, as size grows, instances tend to be more difficult for both methods. However, they confirm the superiority of CBC which solves more instances in the allowed computation time, and is, on average, faster than MODEL. This observations can stimulate further research aimed at obtaining algorithms able to offer better performance for very large instances.

Table 11
Aggregated results for instance sets $M1$, $W1$, and $W2$. $|\mathcal{I}| = 600$, $l = 2$.

SET	MODEL		CBC			
	#OPT	TIME(seconds)	#OPT	TIME(seconds)	#F-CUTS	#P-CUTS
$M1$ ($\lvert\mathcal{K}\rvert = 10$)	10	648.39	10	207.48	2 (3)	10 (240)
$W1$ ($\lvert\mathcal{K}\rvert = 15$)	8	917.55	9	735.79	2 (3)	10 (160)
$W2$ ($\lvert\mathcal{K}\rvert = 20$)	7	1603.13	7	1444.03	3 (4)	10 (143)
Avg.	8.33	1056.36	8.67	795.77	2.33 (3.33)	10 (181)

Table 12
Aggregated results for instance sets $M1$, $Y1$, and $Y2$. $|\mathcal{K}| = 10$, $l = 2$.

SET	MODEL		CBC			
	#OPT	TIME(seconds)	#OPT	TIME(seconds)	#F-CUTS	#P-CUTS
$M1$ ($\lvert\mathcal{I}\rvert = 600$)	10	648.39	10	207.48	2 (3)	10 (240)
$Y1$ ($\lvert\mathcal{I}\rvert = 800$)	6	2177.83	8	1959.93	5 (15)	10 (1424)
$Y2$ ($\lvert\mathcal{I}\rvert = 1000$)	2	3252.77	3	2917.79	2 (4)	10 (425)
Avg.	6	2026.33	7	1695.07	3 (7.33)	10 (696.33)

7. Conclusions and future developments

In this paper we have introduced the Multiple Multidimensional Knapsack Problem with Family-Split Penalties (MMdKFSP), a problem arising in distributed computing resource management. The MMdKFSP is an extension of both the Multi-Knapsack Problem (MKP) and the Multidimensional Knapsack Problem (MdKP), in which items are grouped into families. The profit is not associated with each single item but with the family as a whole. To earn this profit it is necessary to select all the items of the family. A splitting penalty incurs when items from the same family are assigned to different knapsacks. Despite the interest on this problem from an application point of view, it has not yet received attention from the Operational Research community. With this seminal paper we aim at starting a new interesting research topic in the field of Applied Combinatorial Optimization. To solve the problem, we have proposed an ILP Model and a Combinatorial Benders Cuts exact approach named CBC. Several sets of instances with different sizes and layouts have also been introduced as a benchmark to compare the performances of CBC with those of a top level commercial solver dealing with the ILP formulation of the problem. As reported in the detailed analysis of computational results, CBC outperforms the ILP Model both in terms of efficiency and effectiveness, and shows better scalability properties when larger instances are considered. Future developments on this subject may concern the design of algorithms (e.g., exact, metaheuristic or matheuristic approaches) to efficiently handle very large instances in reasonable computational times. The CBC approach proposed in this paper could be generalized to other Applied Combinatorial Optimization problems, showing a similar structure, arising in different contexts such as Logistics, Transportation and Healthcare.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.ejor.2019.07.0521.

References

Adany, R., Feldman, M., Haramaty, E., Khandekar, R., Schieber, B., Schwartz, R., et al. (2013). All-or-nothing generalized assignment with application to scheduling advertising campaigns. In M. Goemans, & J. Correa (Eds.), *Integer programming and combinatorial optimization* (pp. 13–24). Springer.

Akpinar, S., Elmi, A., & Bektaş, T. (2017). Combinatorial benders cuts for assembly line balancing problems with setups. *European Journal of Operational Research, 259*(2), 527–537.

Bai, L., & Rubin, P. A. (2009). Combinatorial benders cuts for the minimum tollbooth problem. *Operations Research, 57*, 1510–1522.

Benders, J. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik, 4*, 238–252.

Bramel, J., & Simchi-Levi, D. (1997). *The logic of logistics: Theory, algorithms, and applications for logistics management*. Springer.

Brucker, P., & Knust, S. (2011). *Complex scheduling* (2nd). Springer.

Cao, J., Lee, D.-H., Chen, J., & Shi, Q. (2010). The integrated yard truck and yard crane scheduling problem: Benders decomposition-based methods. *Transportation Research Part E: Logistics and Transportation Review, 46*(3), 344–353.

Ceri, S., Martella, G., & Pelagatti, G. (1982). Optimal file allocation in a computer network: A solution method based on the knapsack problem. *Computer Networks, 6*(5), 345–357.

Ceselli, A., & Righini, G. (2006). An optimization algorithm for a penalized knapsack problem. *Operations Research Letters, 34*, 394–404.

Chekuri, C., & Khanna, S. (2000). A ptas for the multiple knapsack problem. In Proceedings of the XI ACM SIAM symposium on discrete algorithms (SODA), (pp. 213–222).

Chen, J., Lee, D.-H., & Cao, J. (2012). A combinatorial benders cuts algorithm for the quayside operation problem at container terminals. *Transportation Research Part E: Logistics and Transportation Review, 48*(1), 266–275.

Chen, L., & Zhang, G. (2018). Packing groups of items into multiple knapsacks. *ACM Transactions of Algorithms, 14*(4), 51:1–51:24.

Chen, Y., & Hao, J.-K. (2014). A “reduce and solve” approach for the multiple-choice multidimensional knapsack problem. *European Journal of Operational Research, 239*, 313–322.

Chor, B., & Rivest, R. (1988). A knapsack-type public key cryptosystem based on arithmetic infinite fields. *IEEE Transactions on Information Theory, 34*(5), 901–909.

Chowdhury, N., & Boutaba, R. (2010). A survey of network virtualization. *Computer Networks, 54*(5), 862–876.

Chu, Y., & Xia, Q. (2004). Generating benders cuts for a general class of integer programming problems. In J.-C. Régin, & M. Rueher (Eds.), *Integration of AI and or techniques in constraint programming for combinatorial optimization problems* (pp. 127–141). Springer.

Codato, G., & Fischetti, M. (2006). Combinatorial benders' cuts for mixed-integer linear programming. *Operations Research, 54*, 756–766 .

Côté, J.-F., Dell'Amico, M., & Iori, M. (2014). Combinatorial benders' cuts for the strip packing problem. *Operations Research, 62*(3), 643–661.

da Silva, V. G., Kirikova, M., & Alksnis, G. (2018). Containers for virtualization: An overview. *Applied Computer Systems, 23*(1), 21–27.

Della Croce, F., Pferschy, U., & Scatamacchia, R. (2019). New exact approaches and approximation results for the Penalized Knapsack Problem. *Discrete Applied Mathematics, 253*, 122–135.

Dell'Amico, M., Delorme, M., Iori, M., & Martello, S. (2019). Mathematical models and decomposition methods for the multiple knapsack problem. *European Journal of Operational Research, 274*(3), 886–899.

Ferreira, C., Martin, A., & Weismantel, R. (1996). Solving multiple knapsack problems by cutting planes. *SIAM Journal on Optimization, 6*, 858–877.

Fréville, A. (2004). The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research, 155*(1), 1–21.

Furini, F., Monaci, M., & Traversi, E. (2018). Exact approaches for the knapsack problem with setups. *Computers and Operations Research, 90*, 208–220.

Garey, M., & Johnson, D. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. Freeman.

Geoffrion, A. (1972). Generalized benders decomposition. *Journal of Optimization Theory and Applications, 10*, 237–260.

Han, B., Leblet, J., & Simon, G. (2010). Hard multidimensional multiple choice knapsack problems, an empirical study. *Computers & Operations Research, 37*(1), 172–181.

Hifi, M., Michrafy, M., & Sbihi, A. (2004). Heuristic algorithms for the multiple–choice multidimensional knapsack problem. *Journal of the Operational Research Society, 55*, 1323–1332.

Hooker, J., & Ottoson, G. (2003). Logic-based benders decomposition. *Mathematical Programming, 96*, 33–60.

Kataoka, S. Y. T. (2014). Upper and lower bounding procedures for the multiple knapsack assignment problem. *European Journal of Operational Research, 237*(2), 440–447.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems*. Springer.

Kim, K., & Günther, H.-O. (2007). *Container terminals and cargo systems: Design, operations management, and logistics control issues*. Springer-Verlag.

Lin, E. Y.-H. (1998). A bibliographical survey on some well-known non-standard knapsack problems. *INFOR: Information Systems and Operational Research, 36*(4), 274–317.

Lust, T., & Teghem, J. (2012). The multiobjective multidimensional knapsack problem: A survey and a new approach. *International Transactions in Operational Research, 19*, 495–520.

Mansi, R., Alves, C., Valerio de Carvalho, J., & Hanafi, S. (2013). A hybrid heuristic for the multiple choice multidimensional knapsack problem. *Engineering Optimization, 45*, 983–1004.

Martello, S., & Toth, P. (1990). *Knapsack problems: Algorithms and computer implementations*. Wiley.

Martello, S., & Monaci, M. (2018). Algorithmic approaches to the multiple knapsack assignment problem. *Omega*. doi:10.1016/j.omega.2018.11.013. in press.

Mazumdar, S., & Pranzo, M. (2017). Power efficient server consolidation for cloud data center. *Future Generation Computer Systems, 70*, 4–16.

McLay, L., & Jacobson, S. (2007). Knapsack problems with setups. *Computational Optimization and Applications, 37*(1), 35–47.

Melachrinoudis, E., & Kozanidis, G. (2002). A mixed integer knapsack model for allocating funds to highway safety improvements. *Transportation Research Part A: Policy and Practice, 36*(9), 789–803.

Michel, S., Perrot, N., & Vanderbeck, F. (2009). Knapsack problems with setups. *European Journal of Operational Research, 196*(3), 909–918.

Nauss, M. (1978). The 0–1 knapsack problem with multiple-choice constraints. *European Journal of Operational Research, 2*, 125–131.

Pfeiffer, J. (2007). *Combinatorial auctions and knapsack problems – An analysis of optimization methods*. VDM Verlag.

Pferschy, U., & Scatamacchia, R. (2018). Improved dynamic programming and approximation results for the knapsack problem with setups. *International Transactions in Operational Research, 25*(2), 667–682.

Pisinger, D. (1999). An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research, 114*, 528–541.

Shachnai, H., & Tamir, T. (2001). Polynomial time approximation schemes for class–constrained packing problems. *Journal of Scheduling, 4*, 313–338.

Shojaei, H., Basten, T., Geilen, M., & Davoodi, A. (2013). A fast and scalable multidimensional multiple-choice knapsack heuristic. *ACM Transactions on Design Automation of Electronic Systems, 18*, 51:1–51:24.

Sprent, P., Smeeton, N., Chatfield, C., & Zidek, J. (2000). *Applied nonparametric statistical methods*. Chapman and Hall/CRC.

Sun, X., Ansari, N., & Wang, R. (2016). Optimizing resource utilization of a data center. *IEEE Communications Surveys & Tutorials, 18*(4), 2822–2846.

Taşkin, Z., & Cevik, M. (2013). Combinatorial benders cuts for decomposing IMRT fluence maps using rectangular apertures. *Computers & Operations Research, 40*(9), 2178–2186.

Truyen, E., Van Landuyt, D., Reniers, V., Rafique, A., Lagaisse, B., & Joosen, W. (2016). Towards a container-based architecture for multi-tenant SaaS applications. In *Proceedings of the 15th international workshop on adaptive and reflective middleware* (p. 6). ACM.

Varia, J. (2010). *Architecting for the cloud: Best practices* (pp. 1–21). Amazon Web Services.

Verstichel, J., Kinable, J., De Causmaecker, P., & Vanden Berghe, G. (2015). A combinatorial benders decomposition for the lock scheduling problem. *Computers & Operations Research, 54*, 117–128.

Wang, K., Yang, F., Zhang, Q., Wu, D., & Xu, Y. (2007). Distributed cooperative rate adaptation for energy efficiency in IEEE 802.11-based multihop networks. *IEEE Transactions on Vehicular Technology, 56*(2), 888–898.