Expert Systems With Applications 143 (2020) 113077

Contents lists available at ScienceDirect

Expert Systems With Applications

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

A Lagrangean based solution algorithm for the knapsack problem with setups

Ali Amiri

Department of MSIS, School of Business, Oklahoma State University, Stillwater, OK 74078, USA

ARTICLE INFO

Article history:
Received 18 May 2017
Revised 17 October 2019
Accepted 5 November 2019
Available online 9 November 2019

Keywords:
Knapsack problem with setups
Lagrangean relaxation

ABSTRACT

We consider the knapsack problem with setups which is a generalization of the classical knapsack problem where the items belong to families and an item can be placed in the knapsack only if its family is selected. The problem has received increasing attention by researchers because of its theoretical significance and practical applications related to resource allocation. This paper presents an algorithm based on a Lagrangean relaxation of the problem that produces solutions whose quality can be assessed automatically with the algorithm itself without ever knowing the optimal solutions. We report results of an extensive computational study which show that the method can solve near optimally very large instances of the problem with up to 500 families and 2,000,000 items in reasonable amount of time. This study shows the merit of using the Lagrangean relaxation method to solve the current problem when the constraints to relax are chosen properly.

1. Introduction

One of the extensively studied problems in combinatorial optimization is the classical (binary) knapsack problem (KP). Its popularity stems from its theoretical significance and vast practical applications Martello, Pisinger and Toth (2000). Given a set of items with their profits and weights, the KP involves selecting a subset of items that maximizes the total profit such that the total weight of the selected items does not exceed the knapsack capacity.

The knapsack problem with setups (KPS) is a general extension of the classical knapsack problem (KS) where the items belong to families and an item can be placed in the knapsack only if its family is selected. The selection of a family involves a setup cost and a capacity consumption. The KPS, first introduced in Chajakis and Guignard (1994), has applications in several areas such as production planning and scheduling (see Chebil & Khemakhem, 2015), energy consumption management (see Della Croce, Salassa & Scatamacchia, 2017), and more generally resource allocation (see Della Croce et al., 2017). The KPS has also theoretical significance because of its generalization of the classical knapsack problem.

Chebil and Khemakhem (2015) developed a dynamic programming algorithm to solve optimally the KPS in pseudo-polynomial time. The algorithm employs a novel technique to reduce the storage requirement of the algorithm. Chebil and Khemakhem (2015) reported results of computational tests which show that the algorithm is suitable in solving small size instances with up to 30 families and 10,000 items. Very recently, Della Croce et al. (2017) presented an optimal method for the KPS that reduces the solution space by reducing the range of the number of families which will be part of the optimal solution and dividing the decision variables into two levels, requiring solving simpler ILP problems. Their computational study show that the solution method can solve optimally instances with up to 200 families and 100,000 items. Pferschy & Scatamacchia (2018) proposed an improved dynamic programming algorithm for the KPS by effectively leveraging a method from the literature applied to the Precedence Constraint KP (PCKP). They tested the algorithm using the benchmark datasets used in Chebil and Khemakhem (2015) and Della Croce et al. (2017).

Furini, Monaci and Traversi (2018) studied three alternative ILP formulations of the problem and developed a parallel algorithm, called FMT, which combines dynamic programming and branch and bound. They showed that the algorithm solves efficiently all the benchmark data sets used in Chebil and Khemakhem (2015) and Della Croce et al. (2017). But, they hinted that the algorithm won't scale up to instances with larger number of item families by noting that "it (i.e., their algorithm) is the best algorithm for instances with a small number of classes". This is a serious limitation of the algorithm in handling larger data sets similar to the ones used in our study.

In this paper, we propose a method to solve the problem effectively, but not necessarily optimally. The method is a simple and easy implementation of Lagrangean relaxation technique. The pro-

E-mail address: amiri@okstate.edu

https://doi.org/10.1016/j.eswa.2019.113077
0957-4174/

posed method produces solutions whose quality can be assessed automatically with the method itself without ever knowing the optimal solutions. Indeed, the subgradient technique embedded within Lagrangean relaxation generates upper bounds on the optimal solution values that are used to evaluate the quality of the solutions produced by the proposed method. We report results of an extensive computational study which show that the method can solve near optimally very large instances of the problem with up to 500 families and 2000,000 items, compared to largest instances used in prior studies with only up to 200 families and 100,000 items. In reality, the quality of the solutions seems to improve as the instance size increases.

Similar to the goal of Pferschy & Scatamacchia (2018), our fundamental goal "is to offer a very simple and viable alternative to the algorithms for KPS available in the literature". The contribution of this paper is therefore twofold: First, an alternative method based Lagrangean relaxation is proposed to solve the KPS. Second, besides its simplicity, the method is very powerful and capable of solving very large instances of the problem whose qualities can be assessed by the method itself. The remainder of the paper is organized as follows. Section 2 provides the integer linear programming formulation of KPS. We present the Lagrangean relaxation based approach in Section 3. The following section reports the results of the extensive computational study. The last section concludes the paper with final remarks.

2. Problem formulation

Define m the number of families of items. Each family $j \in \{1, \ldots, m\}$ has a setup cost f_j and a capacity consumption b_j. Each family j contains n_j items. Each item i of family j is characterized by a profit p_{ij} and capacity consumption a_{ij}. The knapsack has a capacity Q. The goal is to maximize the net profit (total profit minus total cost) of the families and their items selected to be included in the knapsack.

Define the decision variables as

$$Y_j = \begin{cases} 1 & if\ family\ j\ is\ placed\ in\ the\ knapsack \\ 0 & otherwiese \end{cases}$$

$$X_{ij} = \begin{cases} 1 & if\ item\ i\ of\ family\ j\ is\ placed\ in\ the\ knapsack \\ 0 & otherwiese \end{cases}$$

Given these definitions, the knapsack problem with setups (*KPS*) can be formulated as model *KPS1* below.

Problem *KPS1*:

$$Z_p = \max \sum_{j=1}^{m} \sum_{i=1}^{n_j} p_{ij} X_{ij} - \sum_{j=1}^{m} f_j Y_j \tag{1}$$

St

$$X_{ij} \le Y_j \quad \forall\ j = 1, \ldots, m\ and\ i = 1, \ldots, n_j \tag{2}$$

$$\sum_{j=1}^{m} \sum_{i=1}^{n_j} a_{ij} X_{ij} + \sum_{j=1}^{m} b_j Y_j \le Q \tag{3}$$

$$X_{ij},\ Y_j \in \{0, 1\} \quad \forall\ j = 1, \ldots, m\ and\ i = 1, \ldots, n_j \tag{4}$$

The objective function maximizes the net profit. Constraint set (2) ensures that an item can be selected for placement in the knapsack only if the family it belongs to is selected for inclusion in the knapsack. Constraint (3) limits the total capacity consumption of the families and items selected to the capacity of the knapsack. Constraints (4) restrict the decision variables to be binary.

3. The solution procedure

We propose a solution approach based on the well-known Lagrangean Relaxation technique. A simple implementation of this technique is very effective in generating good solutions to the KPS. The quality of the solutions is assessed by the technique itself without knowing the optimal solutions. This is an extremely important feature to have, as it is computationally impossible to know the optimal solutions for large instances of the *KPS* similar to the ones used in this paper. An outline of the overall solution method is shown in Fig. 1.

Based on the observation that removing constraint set (2) reduces (*KPS*1) to the binary knapsack problem, we consider the Lagrangean relaxation of (*KPS*1) obtained by dualizing constraint set (2) using nonnegative dual multipliers $\alpha_{ij}\ \forall\ j = 1, \ldots,\ m\ and\ i = 1, \ldots,\ n_j$.

Problem *KSPR*:

$$Z_R = \max \sum_{j=1}^{m} \sum_{i=1}^{n_j} (p_{ij} - \alpha_{ij}) X_{ij} + \sum_{j=1}^{m} (-f_j + \sum_{i=1}^{n_j} \alpha_{ij}) Y_j \tag{5}$$

St

$$\sum_{j=1}^{m} \sum_{i=1}^{n_j} a_{ij} X_{ij} + \sum_{j=1}^{m} b_j Y_j \le Q \tag{6}$$

$$X_{ij},\ Y_j \in \{0, 1\} \quad \forall\ j = 1, \ldots, m\ and\ i = 1, \ldots, n_j \tag{7}$$

The proposed Lagrangean relaxation can potentially produce an upper bound that is tighter than the bound of the linear programming relaxation of (*KSP1*) because our relaxation does not have the integrity property (see Fisher, 1981; Geoffrion, 1974) because the linear programming relaxation of (*KSPR*) does not necessarily have an integer solution.

The Lagrangean problem (*KSPR*) is a (0,1) knapsack problem that can be solved optimally using various specialized methods (see Balas & Zemel, 1980, Martello et al., 2000, Pisinger, 1997). Most of these approaches are based on dynamic programming and are pseudo-polynomial in the capacity of the knapsack. However, we chose to solve the knapsack problem (*KSPR*) using the commercial solver CPLEX as it was found to be very efficient in handling this problem.

The best upper bound to the optimal solution of problem (*KPS*1) is obtained by solving $Z_{UB} = \min_{\alpha \in \Re^+} Z_R(\alpha)$. The Subgradient optimization technique (see Bazaraa & Goode, 1979) is used to optimize the Lagrangean dual. We generate a feasible solution at every iteration using a heuristic embedded in the subgradient optimization procedure. We update the feasible solution of the problem if a better one is generated and report the best feasible solution obtained throughout all the iterations as the final solution to the problem.

The heuristic used to generate feasible solutions is a greedy-based heuristic which uses both the item profits and Lagrangean multipliers to guide the search for better feasible solutions, and can be outlined as fellows.

Procedure Feas:

1. Start with none of the families and their items is placed in the knapsack using the indicators $S_j = 0\ and\ S_{ij} = 0\ \forall j = 1,\ \ldots,\ m\ and\ i = 1,\ \ldots,\ n_j$. Let $\tilde{Q}$ be the remaining capacity of the knapsack; initially $\tilde{Q} = Q$ and the objective function value $\mathcal{P} = 0$.
2. Sort all the items in descending order of $\frac{p_{ij} - \alpha_{ij}}{a_{ij}}$
3. Begin with the item at the top of the sorted list. Say the item has index i^* of family j^*.
4. Check whether placing the item in the knapsack is feasible; that is, check whether the remaining capacity of the knapsack can accommodate the item and its family (if the family has not already been placed earlier in the knapsack). This can be easily

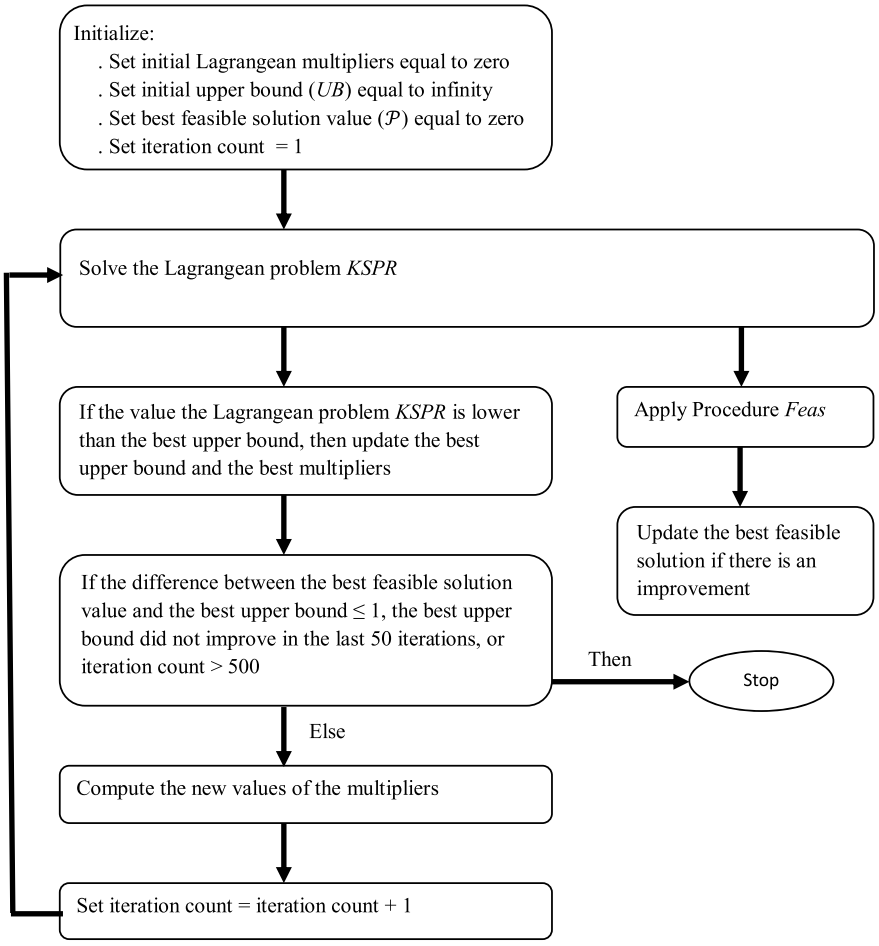

Fig. 1. Illustration of the Lagrangean relaxation-based solution algorithm.

done by checking if $a_{i^*j^*} + (1 - S_{j^*})b_{j^*} \leq \tilde{Q}$. If yes, placed the item and its family (if the family has not already been placed earlier in the knapsack) in the knapsack.

5. Update the objective function value and the remaining capacity of the knapsack; i.e., $\mathcal{P} = \mathcal{P} + p_{i^*j^*} - (1 - S_{j^*})f_{j^*}$; $\tilde{Q} = \tilde{Q} - a_{i^*j^*} - (1 - S_{j^*})b_{j^*}$; if $S_{j^*} = 0$ then *set* $S_{j^*} = 1$. Remove the item i^* from the list.
6. Repeat steps 3–5 until the sorted list is empty or the remaining capacity of the knapsack $\tilde{Q} = 0$.

We have also developed a baseline heuristic to further evaluate the effectiveness of the Lagrangean-based solution approach. One naive way to apply a greedy approach to solve the problem is to admit families and all their items (without exceeding knapsack capacities) in a sequential fashion based on their net profits. Obviously, this way is fast, but it doesn't allow for admitting families and only subsets of their associated items. Here, we propose an enhanced greedy algorithm (called EGA) to solve the problem which overcomes that shortcoming. At each iteration of the algorithm, an attempt is made to identify a family and a subset of its items which, when added to the knapsack, increase the total net profit the most. The selection of the next family and its items to add is based on the following observation. Assume that the items in each family are sorted in non-increasing order of the ratio $\frac{p_{ij}}{a_{ij}}$. We have noticed that, when items within a family are added to the knapsack, the ratio of the (increase in total profit)/(increase in capacity consumption) increases sequentially until it reaches its peak and starts to decrease. After sorting the items of a family j in non-increasing order of the ratio profit per weight (i.e., $\frac{p_{ij}}{a_{ij}}$), we define the peak item i* for family j to be the item that corresponds to the highest value of the peak ratio $P_j = \frac{-f_j + \sum_{i=1}^{i^*} p_{ij}}{b_j + \sum_{i=1}^{i^*} a_{ij}}$. The family and its items corresponding to the highest peak ratio are considered next for addition to the knapsack. The details of the enhanced greedy algorithm are described as follows:

Table 1a
Illustrative example: Parameter values.

Family 1			Family 2		
Item	p_{i1}	a_{i1}	Item	p_{i2}	a_{i2}
1	9668	2697	1	9299	1590
2	7619	2410	2	8320	2326
3	2660	1278	3	5285	1670
4	3388	2189	4	930	5173
5	1235	1900	5	586	3469

EGA:

1. Start with none of the families and their items being placed in the knapsack using the indicators $S_j = 0$ *and* $S_{ij} = 0\ \forall j = 1, \ldots, m$ *and* $i = 1, \ldots, n_j$. Let the P_j be the peak ratio of family j, $\tilde{Q}$ be the remaining capacity of the knapsack; initially $\tilde{Q} = Q$ and the objective function value $\mathcal{P} = 0$.
2. For each family j, sort its items which have not been yet added to the knapsack (i.e., $S_{ij} = 0$) in non-increasing order of the ratio $\frac{p_{ij}}{a_{ij}}$, and compute its peak ratio P_j.
3. Let j^* be the family with the highest peak ratio and i^* its peak item. If the remaining capacity allows, add this family to the

Table 1b
Illustrative example: Step 2 of iteration 1 of *EGA* heuristic.

Family	Profit(p)	Capacity Consum (*cc*)	ratio p/cc	Cumulative profit (cp)	Cumulative Capa. Consum. (ccc)	ratio cp/ccc
	−10,000	5000	−2.000	−10,000	5000	−2.000
	9668	2697	3.584	−332	7697	−0.043
1	7619	2410	3.161	7287	10,107	0.721
	2660	1278	2.081	9947	11,385	0.874
	3388	2189	1.548	13,335	13,574	0.982
	1235	1900	0.650	14,570	15,474	0.942
	−10,000	5000	−2	−10,000	5000	−2.000
	9299	1590	5.848	−701	6590	−0.106
2	8320	2326	3.577	7619	8916	0.855
	5285	1670	3.165	12,904	10,586	1.219
	930	5173	0.180	13,834	15,759	0.878
	586	3469	0.169	14,420	19,228	0.750

Table 1c
Illustrative example: Step 2 of iteration 2 of *EGA* heuristic.

Family	Profit(p)	Capacity Consum (*cc*)	ratio p/cc	Cumulative profit (cp)	Cumulative Capa. Consum. (ccc)	ratio cp/ccc
	−10,000	5000	−2.000	−10,000	5000	−2.000
	9668	2697	3.584	−332	7697	−0.043
1	7619	2410	3.161	7287	10,107	0.721
	2660	1278	2.081	9947	11,385	0.874
	3388	2189	1.548	13,335	13,574	0.982
	1235	1900	0.650	14,570	15,474	0.942
2	930	5173	0.180	930	5173	0.180
	586	3469	0.169	1516	8642	0.175

knapsack (if it was not already added to the knapsack and set $S_{j^*} = 1$), and add items in the sorted list of this family (which were not already added to the knapsack and set $S_{ij^*} = 1$ for those items) up to the peak item i^* without exceeding the knapsack capacity. Update $\tilde{Q}$ and $\mathcal{P}$ accordingly.

4. Repeat steps 2 and 3 until the total net profit cannot be improved any further and/or the remaining capacity cannot accommodate adding more families and items.

We illustrate the process of applying the procedure *EGA* using an example with two families and five items in each family, where $f_1 = f_2 = 10,000$ and $b_1 = b_2 = 5000$. The profits and capacity consumptions for the items are shown in Table 1(a). The knapsack capacity is 20,000. The details of applying step 2 of iteration 1 to the example are shown in Table 1(b). The peak ratio for the first family is 0.982 corresponding to peak item 4. The peak ratio for the second family is 1.219 corresponding to peak item 3. In step 3, $j^* = 2$ because the peak ratio of family 2 is greater than the peak ratio of family 1; hence, we consider adding first family 2 and its first three items to the knapsack as allowed by the knapsack capacity. We add family 2 and its first three items to the knapsack. At the end of step 3 of iteration 1, the total profit is 12,904 and total capacity consumption is 10,586. The details of applying step 2 of iteration 2 are shown in Table 1(c) for both families. The peak ratio of family 1 is now 0.982 and the peak ratio of family 2 is 0.180. In step 3 of iteration 2, $j^* = 1$ because the peak ratio of family 1 is greater than the peak ratio of family 2; hence, we consider adding next family 1 and its first four items to the knapsack as allowed by the knapsack capacity. Based on the remaining capacity (20,000–10,586), we add family 1 and items 1 and 3 to the knapsack. At the end of this iteration 2, the total profit is 15,232 and total capacity consumption is 19,561. Since the remaining knapsack capacity (20,000–19,561) cannot accommodate any more items, the algorithm terminates. The solution obtained by this algorithm with a total net profit of 15,232 is confirmed to be optimal using CPLEX.

Table 2
Computational results for the benchmark instances.

m	n	Gap_{Lag}	Gap_{Opt}	Gap_{dual}	Time(s)
5	500	0.866	0.163	0.667	5
	1000	1.164	0.514	0.619	5
	2500	1.242	0.805	0.413	5
	5000	0.755	0.321	0.424	8
	10,000	1.136	0.946	0.181	17
10	500	0.573	0.169	0.390	5
	1000	1.181	0.076	1.054	5
	2500	1.280	0.242	0.977	5
	5000	0.536	0.315	0.217	8
	10,000	0.442	0.303	0.136	16
20	500	0.157	0.091	0.066	4
	1000	0.499	0.240	0.254	5
	2500	0.579	0.162	0.406	5
	5000	0.671	0.382	0.282	8
	10,000	0.770	0.311	0.445	16
30	500	0.089	0.039	0.049	4
	1000	0.374	0.146	0.226	5
	2500	0.271	0.137	0.132	5
	5000	0.193	0.147	0.045	7
	10,000	0.288	0.247	0.040	17

4. Computational study

We run a computational study to assess the effectiveness of the proposed algorithm in solving the knapsack problem with setup (KPS). We coded the Lagrangean relaxation based solution procedure in Visual Studio.Net and run it on Intel Core i7 with 3.6 GHz and 32 GB of RAM. The procedure was applied to benchmark instances used in prior research studies (see Chebil & Khemakhem, 2015), Della Croce et al., 2017), Pferschy & Scatamacchia, 2018; Furini et al., 2018) and to additional sets of instances which were generated randomly, but systematically according to the scheme described in Chebil and Khemakhem (2015), and Della Croce et al. (2017).

Table 3
Computational results for the large instances.

			EGA					Lagrangean			
m	n	$Gap_{Blanket}$	Capacity Utilization	%Family Admitted	%Items Admitted	Time (s)	Gap_{Lag}	Capacity Utilization	%Family Admitted	%Items Admitted	Time (s)
20	100,000	5.21	94.91	40.00	40.00	1	0.96	99.92	44.50	42.59	186
20	200,000	5.01	95.12	40.00	40.00	1	0.90	99.82	44.00	42.33	364
20	300,000	4.95	95.17	40.00	40.00	2	0.99	99.75	44.00	42.30	604
20	400,000	4.92	95.21	40.00	40.00	3	1.21	99.74	44.00	42.30	816
20	500,000	4.88	95.25	40.00	40.00	5	1.38	99.75	44.00	42.26	1112
20	600,000	4.86	95.27	40.00	40.00	6	1.11	99.76	44.00	42.29	1405
20	700,000	4.85	95.28	40.00	40.00	7	1.39	99.76	44.00	42.26	1622
20	800,000	4.84	95.29	40.00	40.00	8	0.92	99.77	44.00	42.26	1819
20	900,000	4.83	95.30	40.00	40.00	9	1.35	99.76	44.00	42.31	1994
20	1000,000	4.82	95.31	40.00	40.00	10	0.99	99.77	44.00	42.31	2130
40	100,000	3.03	96.97	41.00	41.00	1	0.54	99.84	43.00	42.36	178
40	200,000	2.80	97.21	41.00	41.00	2	0.58	99.79	42.75	42.21	370
40	300,000	2.67	97.33	41.00	41.00	4	0.53	99.92	43.00	42.26	556
40	400,000	2.58	97.43	41.00	41.00	6	0.46	99.86	42.75	42.16	760
40	500,000	2.54	97.47	41.00	41.00	7	0.42	99.88	42.75	42.15	954
40	600,000	3.11	96.91	40.75	40.75	9	0.44	99.88	42.75	42.13	1151
40	700,000	3.09	96.93	40.75	40.75	11	0.36	99.88	42.75	42.13	1325
40	800,000	3.08	96.94	40.75	40.75	13	0.34	99.88	42.75	42.14	1514
40	900,000	3.05	96.97	40.75	40.75	15	0.51	99.89	42.75	42.12	1788
40	1000,000	3.06	96.96	40.75	40.75	18	0.40	99.89	42.75	42.12	1818
60	100,000	1.99	97.99	41.50	41.51	1	0.39	99.96	43.00	42.46	174
60	200,000	1.72	98.26	41.50	41.50	4	0.28	99.90	42.67	42.27	357
60	300,000	1.59	98.39	41.50	41.50	6	0.41	99.96	42.83	42.30	553
60	400,000	1.88	98.13	41.33	41.34	8	0.26	99.86	42.50	42.15	752
60	500,000	1.83	98.18	41.33	41.33	10	0.30	99.91	42.67	42.17	938
60	600,000	1.80	98.21	41.33	41.33	13	0.39	99.85	42.50	42.10	1165
60	700,000	1.78	98.23	41.33	41.33	15	0.41	99.86	42.50	42.11	1298
60	800,000	1.75	98.26	41.33	41.33	18	0.31	99.86	42.50	42.10	1538
60	900,000	1.72	98.29	41.33	41.33	21	0.39	99.87	42.50	42.09	1770
60	1000,000	1.71	98.30	41.33	41.33	24	0.28	99.87	42.50	42.09	1921
80	100,000	1.31	98.67	41.88	41.88	2	0.19	99.95	42.75	42.46	175
80	200,000	1.84	98.17	41.50	41.50	5	0.21	99.98	42.63	42.33	364
80	300,000	1.70	98.31	41.50	41.50	8	0.19	99.97	42.50	42.25	531
80	400,000	1.88	98.12	41.38	41.38	10	0.16	99.96	42.50	42.22	738
80	500,000	1.82	98.18	41.38	41.38	14	0.23	99.98	42.50	42.20	953
80	600,000	1.77	98.22	41.38	41.38	16	0.23	99.99	42.50	42.19	1153
80	700,000	1.74	98.26	41.38	41.38	20	0.29	99.98	42.50	42.18	1367
80	800,000	1.70	98.30	41.38	41.38	23	0.18	99.99	42.50	42.18	1804
80	900,000	1.67	98.33	41.38	41.38	26	0.21	100.00	42.50	42.16	2418
80	1000,000	1.65	98.34	41.38	41.38	30	0.25	100.00	42.50	42.15	2880

4.1. Results from benchmark data sets

Prior research studies (Chebil & Khemakhem, 2015), Della Croce et al., 2017; Pferschy & Scatamacchia, 2018;, and Furini et al., 2018) used a set of 20 categories of 10 problem instances in each category. The instances are strongly correlated and generated randomly as follows. The various values of the number of families are 5, 10, 20, and 30. Five possible values of total number of items in all families (n) are used, namely {500, 1000, 2500, 5000, 10,000}. The number of items in each family n_j is in the range [k-k/10, $k+k$/10] where $k=n/m$. The setup cost and capacity consumption of each family j are $f_j = -e_1(\sum_{i=1}^{n_j} p_{ij})$ and $b_j = -e_1(\sum_{i=1}^{n_j} a_{ij})$, respectively, where e_1 is drawn from the uniform distribution [0.15, 0.25]. The item profit p_{ij} is randomly generated in [10,100]. The item capacity consumption is set as $c_{ij}=p_{ij}+10$. The knapsack capacity is set as $Q = 0.5 * \sum_{j=1}^{m}\sum_{i=1}^{n_j} a_{ij}$. The values of these parameters for all 200 instances can be found at https://sites.google.com/site/chebilkh/knap sack-problem-with-setup.

Table 2 shows the results of solving the benchmark instances using the proposed Lagrangean relaxation based solution procedure. The results are averages over the 10 instances of each category. Column 1 displays the Lagrangean gap (Gap_{Lag}) between the best feasible solution value ($\mathcal{P}$) and upper bound (UB) generated by the procedure and computed as follows: $Gap_{Lag} = 100 * (UB - \mathcal{P})/UB$. Column 4 displays the optimality gap (Gap_{opt}) between the best feasible solution value ($\mathcal{P}$) generated by the procedure and the optimal solution value ($\mathcal{P}^*$) generated by CPLEX and computed as follows: $Gap_{opt} = 100 * (\mathcal{P}^* - \mathcal{P})/\mathcal{P}^*$. In order to assess the tightness of the upper bound (UB) generated by the proposed procedure, column 5 displays the duality gap (Gap_{dual}) between the upper bound and the optimal solution value ($\mathcal{P}^*$) and computed as follows: $Gap_{dual} = 100 * (UB - \mathcal{P}^*)/\mathcal{P}^*$. Column 6 displays the CPU time taken by the Lagrangean relaxation based solution procedure. The Lagrangean gap Gap_{Lag} varied between 0.182% and 2.669%, with an average of 0.653% over all 200 instances. The optimality gap Gap_{opt} varied between 0.085% and 1.961%, with an average of 0.288% over all 200 instances. The duality gap Gap_{dual} varied between 0.092 and 2.223%, with an average of 0.351% over all 200 instances. The proposed solution procedure is quite fast in solving the problem, with an average CPU time of 7.75 s.

4.2. Results from larger generated data

To further analyze the performance of the proposed solution approach, we generated significantly larger problem instances following the same scheme used in Chebil and Khemakhem (2015), and Della Croce et al. (2017) as described above. We conducted two computational studies. In the first one, we varied the number of families (m) between 20 and 80 and the total number of items (n) between 100,000 and 1000,000. In the second study, we

Table 4
Computational results for the very large instances.

			EGA					Lagrangean			
m	n	$Gap_{Blanket}$	Capacity Utilization	%Family Admitted	%Items Admitted	Time (s)	Gap_{Lag}	Capacity Utilization	%Family Admitted	%Items Admitted	Time (s)
100	100,000	1.38	98.63	41.90	41.90	2	0.17	100.00	42.90	42.56	171
	200,000	1.00	99.01	41.90	41.90	6	0.17	99.97	42.70	42.38	368
	300,000	1.06	98.96	41.80	41.80	9	0.17	99.97	42.60	42.29	559
	400,000	1.19	98.81	41.70	41.70	13	0.20	100.00	42.60	42.27	761
	500,000	1.11	98.89	41.70	41.70	16	0.23	100.00	42.60	42.25	899
	600,000	1.30	98.69	41.60	41.60	20	0.21	100.00	42.60	42.23	1140
	700,000	1.27	98.73	41.60	41.60	23	0.22	99.96	42.50	42.19	1330
	800,000	1.23	98.77	41.60	41.60	27	0.17	99.96	42.50	42.18	1521
	900,000	1.19	98.81	41.60	41.60	30	0.22	99.98	42.50	42.16	1722
	1000,000	1.17	98.83	41.60	41.60	34	0.19	99.97	42.50	42.15	1945
200	100,000	0.43	99.56	42.55	42.55	5	0.10	99.98	42.90	42.76	173
	200,000	0.57	99.44	42.25	42.25	11	0.13	99.99	42.70	42.53	355
	300,000	0.43	99.57	42.20	42.20	18	0.11	99.96	42.55	42.40	543
	400,000	0.51	99.47	42.10	42.10	24	0.09	99.97	42.45	42.33	759
	500,000	0.65	99.35	42.00	42.00	31	0.07	99.98	42.40	42.29	928
	600,000	0.71	99.29	41.95	41.95	37	0.08	99.99	42.40	42.27	1089
	700,000	0.66	99.34	41.95	41.95	44	0.06	99.97	42.35	42.24	1379
	800,000	0.71	99.28	41.90	41.90	51	0.05	99.98	42.30	42.21	1536
	900,000	0.68	99.31	41.90	41.90	58	0.07	99.98	42.30	42.21	1735
	1000,000	0.64	99.35	41.90	41.90	65	0.06	99.99	42.30	42.19	1939
300	100,000	0.37	99.63	42.73	42.77	8	0.03	99.97	42.93	42.92	175
	200,000	0.35	99.65	42.43	42.47	17	0.06	99.98	42.70	42.64	365
	300,000	0.38	99.62	42.33	42.33	26	0.04	100.00	42.57	42.50	531
	400,000	0.48	99.51	42.20	42.20	35	0.05	100.00	42.53	42.43	758
	500,000	0.43	99.58	42.17	42.17	45	0.06	99.99	42.47	42.37	963
	600,000	0.38	99.63	42.17	42.17	55	0.08	99.99	42.47	42.35	1163
	700,000	0.36	99.64	42.13	42.14	65	0.10	100.00	42.47	42.32	1374
	800,000	0.29	99.71	42.13	42.14	75	0.07	99.98	42.37	42.28	1558
	900,000	0.27	99.73	42.13	42.13	85	0.08	99.98	42.37	42.26	1750
	1000,000	0.38	99.62	42.07	42.07	95	0.06	99.98	42.33	42.25	1874
400	100,000	0.33	99.67	42.95	42.95	10	0.03	99.99	43.15	43.10	174
	200,000	0.26	99.74	42.63	42.63	22	0.05	99.99	42.83	42.75	358
	300,000	0.27	99.73	42.48	42.48	34	0.05	99.99	42.68	42.60	544
	400,000	0.25	99.75	42.38	42.38	46	0.04	99.98	42.55	42.49	748
	500,000	0.29	99.71	42.30	42.30	59	0.04	99.99	42.50	42.44	955
	600,000	0.30	99.70	42.25	42.25	72	0.05	100.00	42.48	42.39	1142
	700,000	0.24	99.76	42.25	42.25	85	0.06	99.98	42.43	42.36	1336
	800,000	0.28	99.72	42.20	42.20	98	0.03	100.00	42.40	42.33	1499
	900,000	0.35	99.65	42.15	42.15	111	0.05	100.00	42.40	42.32	1741
	1000,000	0.30	99.70	42.15	42.15	125	0.06	99.99	42.38	42.29	1936
500	100,000	0.22	99.78	43.12	43.12	13	0.02	99.97	43.22	43.20	178
	200,000	0.22	99.78	42.74	42.74	27	0.04	99.99	42.90	42.84	363
	300,000	0.21	99.79	42.58	42.58	42	0.05	99.99	42.76	42.69	558
	400,000	0.22	99.78	42.46	42.46	57	0.03	99.99	42.60	42.56	748
	500,000	0.20	99.80	42.40	42.40	73	0.04	99.99	42.56	42.49	958
	600,000	0.24	99.77	42.34	42.34	89	0.03	99.99	42.50	42.45	1142
	700,000	0.24	99.76	42.30	42.30	105	0.02	99.99	42.44	42.40	1335
	800,000	0.21	99.79	42.28	42.28	121	0.02	99.99	42.40	42.37	1521
	900,000	0.34	99.66	42.20	42.20	137	0.02	100.00	42.40	42.36	1665
	1000,000	0.28	99.72	42.20	42.20	153	0.04	100.00	42.40	42.33	1934
100	1500,000	1.09	98.91	41.60	41.60	53	0.26	99.98	42.50	42.12	3164
200	1500,000	0.76	99.24	41.80	41.80	99	0.07	99.99	42.25	42.14	3011
300	1500,000	0.40	99.60	42.00	42.00	146	0.04	99.99	42.23	42.18	2709
400	1500,000	0.32	99.68	42.08	42.08	191	0.04	100.00	42.30	42.23	3025
500	1500,000	0.20	99.80	42.16	42.16	236	0.04	99.98	42.30	42.25	2934
100	2000,000	1.05	98.95	41.60	41.60	74	0.19	99.98	42.50	42.11	4341
200	2000,000	0.70	99.30	41.80	41.80	134	0.08	99.99	42.25	42.12	4085
300	2000,000	0.54	99.45	41.90	41.90	197	0.03	99.99	42.20	42.14	4074
400	2000,000	0.28	99.72	42.05	42.05	258	0.06	99.99	42.25	42.18	4092
500	2000,000	0.24	99.76	42.10	42.10	318	0.03	99.99	42.26	42.21	4070

varied the number of families (m) between 100 and 500 and the total number of items (n) between 100,000 and 2000,000. Ten instances were generated of each of the 110 categories/combinations of m and n, resulting in a total of 1000 large instances. In addition to the Lagrangean gap (Gap_{Lag}) and time, we display the knapsack capacity utilization. The results reported in Tables 3 and 4 are averages over the 10 instances of each category.

The proposed Lagrangean relaxation based solution procedure is very effective in tackling these large instances of the problem. The performance of the Lagrangean relaxation-based solution procedure is significantly better than that of the EGA heuristic. Indeed, based on the results for the very large problem instances reported in Table 4, the Lagrangean gap Gap_{Lag} varied between 0.01% and 0.46%, with an average of 0.08 over all 600 instances. Whereas, the gaps for the EGA heuristic (using the upper bound values generated by the Lagrangean procedure) over the same instances varied between 0.19% and 1.38%, with an average of 0.54%. Again, the gaps provide an objective assessment of the quality of the solutions gen-

Table 5
Computational results with relatively large number of families.

			EGA					Lagrangean			
m	n	$Gap_{Blanket}$	Capacity Utilization	%Family Admitted	%Items Admitted	Time (s)	Gap_{Lag}	Capacity Utilization	%Family Admitted	%Items Admitted	Time (s)
100	5000	1.55	98.44	43.90	43.90	1	0.32	99.96	44.80	44.58	15
	10,000	1.73	98.26	43.00	43.00	1	0.20	99.97	43.90	43.76	30
	15,000	1.15	98.84	42.90	42.90	1	0.44	99.96	43.80	43.46	47
	20,000	0.76	99.24	42.90	42.90	1	0.59	99.97	43.80	43.32	62
200	5000	0.58	99.42	45.45	45.45	1	0.12	99.95	45.75	45.68	15
	10,000	0.53	99.46	44.30	44.30	1	0.14	99.96	44.60	44.52	32
	15,000	0.47	99.52	43.80	43.80	1	0.22	99.98	44.20	44.03	48
	20,000	0.78	99.21	43.40	43.40	1	0.13	99.99	43.85	43.75	63
300	5000	0.39	99.64	44.00	46.24	1	0.18	99.97	46.90	46.49	16
	10,000	0.45	99.55	44.37	44.92	1	0.11	99.96	45.03	45.13	32
	15,000	0.35	99.65	44.37	44.37	1	0.11	99.98	44.60	44.52	48
	20,000	0.26	99.74	43.50	44.07	1	0.15	99.98	44.17	44.21	65
400	5000	0.45	99.59	44.65	46.86	1	0.15	100.00	47.65	47.21	16
	10,000	0.17	99.83	45.55	45.55	1	0.07	99.96	45.65	45.62	30
	15,000	0.29	99.71	44.00	44.75	1	0.07	99.99	44.88	44.90	48
	20,000	0.31	99.68	44.38	44.38	2	0.09	100.00	44.58	44.51	64
500	5000	0.24	99.79	47.68	47.68	1	0.06	99.99	47.92	47.88	15
	10,000	0.31	99.69	45.98	45.98	1	0.06	100.00	46.18	46.13	32
	15,000	0.25	99.75	45.22	45.22	1	0.07	100.00	45.40	45.34	48
	20,000	0.24	99.75	44.74	44.74	2	0.09	100.00	44.94	44.86	64

erated by the proposed procedures without even knowing the optimal solutions. This is really one of the very powerful features of the Lagrangean relaxation technique that generates automatically the upper bound which is used as an overestimate of the unknown optimal solution value. The success of the implementation of this technique to the KPS is achieved by generating good feasible solutions and tight upper bounds, resulting in small Lagrangean gaps between the two. The knapsack capacity is almost fully utilized. The CPU times are quite reasonable given the size of the instances. CPLEX failed to generate even feasible solutions to the much larger instances.

We also wanted to explore the effect of higher ratio of (number of Families/number of items) (i.e., $\frac{m}{\sum_{j=1}^{m} n_j}$) on the performance of the Lagrangean based solution procedure. This ratio varied between 0.0005 and 0.06 for the benchmark datasets, and between 0.00002 and 0.005 for the other larger randomly generated datasets. For this exploration purpose, we generated 200 additional random datasets where the number of families varies between 100 and 500 and the total number of items varies between 5000 and 20,000; resulting in the above ratio varying between 0.005 and 0.1. The effects of higher values of that ratio on the performance of the solution procedures are reported on Table 5. The Lagrangean based solution procedure continues to perform quite well. Indeed, the Lagrangean gap Gap_{Lag} varied between 0.00044% and 0.92%, with an average of 0.17 over all 200 instances. The performance of the Lagrangean relaxation-based solution procedure is significantly better than that of the EGA heuristic. Indeed, the gaps for the EGA heuristic (using the upper bound values generated by the Lagrangean procedure) over the same 200 instances varied between 0.00044% and 2.25%, with an average of 0.56%.

Based on the results of the computational study, the following implications are worth mentioning. The proposed method based on the Lagrangean relaxation technique to solve the KPS offers the following benefits: (i) the method produces solutions whose quality can be assessed automatically with the method itself without ever knowing the optimal solutions, and (ii) the method is effective and scalable in solving very large instances of the problem. A third benefit is of theoretical nature: In general, a potential weakness of the use of the Lagrangean relaxation technique to solve integer programs is its slowness or inability to converge. Interestingly, our application of the Lagrangean technique doesn't suffer from this weakness. Indeed, we chose properly the constraints to relax/dualize. In general, this choice is guided by two factors: (i) ease of solving the relaxed problem, and (ii) potential of generating good upper bounds. In our case, there are two obvious and different Lagrangean relaxations for the KPS: The first one consists of relaxing constraint set (3) which results in a Lagrangean problem which is very easy to solve, but has the integrality property (i.e., the best upper bound generated by this relaxation cannot be better than the LP bound of the original problem); the second relaxation for the KPS is the one we adopted in this paper (i.e., relaxing constraint set (2)), this relaxation results in a Lagrangean problem which is less easy to solve, but does not have the integrality property.

5. Conclusion

In this paper, we studied the knapsack problem with setups where the items belong to families and an item can be placed in the knapsack only if its family is selected. We proposed an algorithm based on a Lagrangean relaxation of the problem to solve the problem effectively. The algorithm produces solutions whose quality can be assessed automatically with the method itself without ever knowing the optimal solutions. We run an extensive computational study using a benchmark data set and very large data sets with up to 500 families and 2000,000 items. These sets are much larger than instances with up to 200 families and 100,000 items used in prior studies. The computational results show that the algorithm can solve near optimally the problem in reasonable amount of time. In reality, the quality of the solutions seems to improve as the problem size increases.

As a direction for future research, it would be interesting to apply the Lagrangean relaxation technique to the extended multiple knapsack problem with setups (MKPS) suggested by Chebil and Khemamkhem (2015) where there are multiple knapsacks (rather than just one) that the families and items can be assigned to. We are currently working on this extension. Another future research area (Furini et al., 2018) includes the study of some general versions of the KPS where (i) lower and upper limits are imposed on the total capacity consumptions of admitted families, and (ii) there are restrictions related to incompatibilities among families/items.

Declaration of Competing Interests

I declare that there is no conflict of interest involved in authoring/published my above paper.

Credit author statement

Ali Amiri: All the work.

References

Balas, E., & Zemel, E. (1980). An algorithm for large Zero–One knapsack problems. *Operations Research, 28*, 1130–1154.

Bazaraa, M. S., & Goode, J. J. (1979). A survey of various tactics for generating Lagrangean multipliers in the context of Lagrangean duality. *European Journal of Operational Research, 3*, 322–328.

Chajakis, E. D., & Guignard, M. (1994). Exact algorithms for the setup knapsack problem. *INFOR, 32*, 124–142.

Chebil, K., & Khemakhem, M. (2015). A dynamic programming algorithm for the knapsack problem with setup. *Computers & Operations Research, 64*, 40–50.

Della Croce, F., Salassa, F., & Scatamacchia, R. (2017). An exact approach for the 0–1 knapsack problem with setups. *Computers & Operations Research, 80*, 61–67.

Fisher, M. (1981). The Lagrangean relaxation method for solving integer programming problems. *Management Science, 27*, 1–18.

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

Geoffrion, A. (1974). Lagrangean relaxation for integer programming. *Mathematical Programming Study, 2*, 82–114.

Martello, S., Pisinger, D., & Toth, P. (2000). New trends in exact algorithms for the 0–1 knapsack problems. *European Journal of Operational Research, 123*, 325–332.

Pferschy, U., & Scatamacchia, R. (2018). Improved dynamic programming and approximation results for the knapsack problem with setups. *Intl. Trans. in Op. Res., 25*(2), 667–682.

Pisinger, D. (1997). A minimal algorithm for the 0-1 knapsack problem. *Operations Research, 45*, 758–767.