International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:6, 2010

# The knapsack sharing problem: a tree search exact algorithm

Mhand Hifi and Hedi Mhalla

*Abstract*—In this paper, we study the knapsack sharing problem, a variant of the well-known NP-Hard single knapsack problem. We investigate the use of a tree search for optimally solving the problem. The used method combines two complementary phases: a reduction interval search phase and a branch and bound procedure one. First, the reduction phase applies a polynomial reduction strategy; that is used for decomposing the problem into a series of knapsack problems. Second, the tree search procedure is applied in order to attain a set of optimal capacities characterizing the knapsack problems. Finally, the performance of the proposed optimal algorithm is evaluated on a set of instances of the literature and its runtime is compared to the best exact algorithm of the literature.

*Keywords*—branch and bound, combinatorial optimization, knapsack, knapsack sharing, heuristics, interval reduction.

## I. Introduction

In this paper, we investigate the use of a tree search exact algorithm for solving the *knapsack sharing problem* (KSP). KSP is a variant of the well-known knapsack problem (KP), an NP-hard combinatorial optimization problem. An instance of KSP is characterized by a capacity $c$, a set $\mathcal{N}$ of $n$ items, where each item $j \in \mathcal{N}$ has a weight $w_j$, a profit $p_j$ and a demand $d_j$. Moreover, the set $\mathcal{N}$ of items denotes a collection of $m$ disjoint classes of items; that is, $\mathcal{N} = \cup_{i=1}^{m} J_i$ and $\forall p \in \mathcal{N},\ q \in \mathcal{N}, p \neq q, J_p \cap J_q = \emptyset$. The objective of the problem is to determine the subset of items satisfying the capacity constraint $c$ as to maximize the minimal value of a set of linear functions. The KSP can be stated as follows:

$$(KSP)\begin{cases} \text{Maximize} & \min\limits_{1\leq i\leq m} \left\{ \sum\limits_{j\in J_i} p_j x_j \right\} \\ \text{Subject to} & \sum\limits_{j\in\mathcal{N}} w_j x_j \leq c \\ & x_j \in \{0,1\},\ \forall j \in \mathcal{N} \end{cases}$$

where $x_j,\ j \in \mathcal{N}$, denotes the binary decision variable such that $x_j = 1$ if the item $j$ is in the solution set, $x_j = 0$ otherwise. Without any loss of generality, we assume that $\sum_{j\in\mathcal{N}} w_j > c$, all classes are indexed from $J_1$ to $J_m$ while the elements of class $J_i$, for $i = 1, \ldots, m$, are indexed from 1 to $|J_i|$. We also assume that $w_j,\ p_j,\ d_j$, and $c$ are all nonnegative integers.

The KSP has a wide range of commercial applications (see Brown [1] and Tang [12]) and, the binary version of the problem is NP-hard since it is a generalization of the single knapsack (Martello and Toth [9], [10]). It is classified as $KSP(Bn/m/1)$ (see Yamada and Futakawa [13], and Hifi and Sadfi [3], [5]), which means that we have $n$ items of binary ($B$) type divided in $m$ classes with one constraint.

In this paper, we propose an exact tree search algorithm for the KSP. The proposed algorithm can be viewed as a two-phase approach which combines a reduction interval search and a branch and bound procedure. The first phase of the algorithm consists in decomposing the KSP into a series of KPs. The provided capacities associated o the KPs are reached by using a heuristic. The second phase of the algorithm tries to determine an attainable optimal set of capacities; that is a set of capacities reaching the final optimal solution. Both phases are complementary and essential to the success of the algorithm, since the second phase is applied particularly for stabilizing the set of the capacities given by the heuristic.

The remainder of the paper is organized as follows. First, Section II presents a brief literature survey of the knapsack sharing problems. Second, the main steps of the exact tree search algorithm is described in Section III.Third, Section IV evaluates, on a set of instances taken from the literature, the performance of the exact tree search algorithm, compared to the more performent exact algorithm of the literature (when considering these instances). Finally, in conclusion, we summarize the main results of the paper.

## II. Related literature

Because of its NP-hardness, KSP has received little attention. A very few published papers addressing the problem of sharing a given capacity are available. The first KSP, namely max-min allocation problem, has been widely studied (Brown [2], Kuno *et al.* [7], Luss [8], Pang and Yu [11], Tang [12]). Different exact and approximate approaches have been designed especially for this problem. For the particular continuous KSP, Kuno *et al.* [7] have proposed a linear time solution algorithm. Yamada and Futakawa [13] proposed another algorithm for the continuous $KSP(Cn/m/1)$.

The KSP has been addressed by Yamada and Futakawa [13] who extended the heuristic approach tailored to the $KSP(Cn/m/1)$ to $KSP(Bn/m/1)$ and Yamada *et al.* [14] who developed several exact algorithms based upon branch and bound procedures or binary search method. The authors indicated that the binary search approach outperformed the branch and bound algorithm. Hifi *et al.* [4] proposed an approximate algorithm based upon tabu search and showed that the method performed well for correlated and uncorrelated problem instances. Hifi and Sadfi [3] designed a dynamic programming algorithm in which the original problem is decomposed into a series of single knapsack problems. The

M. Hifi is a Professor at Université de Picardie Jules Verne, Equipe ROAD, UR MIS, 33 rue Saint Leu, 80000 Amiens, France (*hifi@u-picardie.fr*).
H. Mhalla is an Associate Professor at Université de Picardie Jules Verne, Equipe ROAD, UR MIS, 33 rue Saint Leu, 80000 Amiens, France. (*hedi.mhalla@u-picardie.fr*)

algorithm has a good behavior especially for the problem instances containing an important set of classes. Hifi *et al.* [5] have proposed a new version of the dynamic programming algorithm already proposed in [3] in order to accelerate the used search process. The computational results showed that the new version of the algorithm was able to improve the performance of the last version of the algorithm especially for the instances containing more than ten classes and, fails in general when the instance problem contains less than or equal to ten classes.

In this paper, we propose an exact algorithm especially when the density of the classes is small, i.e., the number of the considered classes is small or equal to ten. The main idea of the paper is to replace the dynamic programming resolution (Hifi *et al.* [4]) by a specialized tree-search algorithm; that is a branch and bound procedure combined with an interval reduction search. In order to make the paper more clearer, we repeat some points already developed in Hifi *et al.* [3], [4], [5] and for the rest of the paper, we adopt the following notations.

- $S(\mathcal{P})$ : represents a feasible solution of the problem $\mathcal{P}$ with value $VS(\mathcal{P})$.
- $Opt(\mathcal{P})$ : denotes an optimal solution of the problem $\mathcal{P}$ with value $VO(\mathcal{P})$.

## III. A TREE SEARCH EXACT ALGORITHM

We first describe the decomposition of the original problem into a series of knapsack problems (see Hifi and Sadfi [3]). We then show the calculus of the initial set of capacities. Finally, we describe the main steps of the proposed tree search exact algorithm.

### A. Reduction of the KSP

The KSP can be reduced to a series of knapsack problems (see Hifi and Sadfi [3]). By applying the same strategy, and in order to clarify the contents of the paper, we summarize some points already presented in Hifi and Sadfi [3]. Let $SK_{J_i}^{\bar{c}_i},\ i = 1, \ldots, m$, be the auxiliary problems associated to KSP. The series of the knapsack problems $SK_{J_i}^{\bar{c}_i},\ i = 1, \ldots, m$, can be stated as follows:

$$SK_{J_i}^{\bar{c}_i} \begin{cases} \max & \sum_{j \in J_i} p_j x_j \\ \text{Subject to} & \sum_{j \in J_i} w_j x_j \leq \bar{c}_i \\ & x_j \in \{0,1\}, \text{ for } j \in J_i, \end{cases}$$

where $\bar{c}_i$ is a nonnegative integer satisfying $\bar{c}_i \leq c,\ \forall i \in \{1, \ldots, m\}$ and, each $SK_{J_i}^{\bar{c}_i}$ $-$represents a knapsack problem$-$ is associated with each specified class $J_i$, for $i = 1, ..., m$, and with capacity $\bar{c}_i$.

### B. An initial set of capacities-based solution

The initial set of capacities-based solution uses the so-called *critical elements* (Hifi *et al.* [4], [5]), where a critical element of each class is the one that is able to decompose the current class into two complementary areas-parts: the *right-critical* and the *left-critical* areas. The initial solution, associated to the set of the capacities provided, is given as follows:

1) Consider a portion of each $S_i$, for $i = 1, \ldots, m$, where $S_i$ denotes the binary representation of the $i$-th class. Suppose that $S_i(k),\ k \leq |J_i|$, is a *critical element* of the $i$-th class.
2) Fix all items of the left-critical region (of each class) to "one" and consider that all elements of the right-critical region as "free".

Note that the obtained solution (steps 1 and 2 above) represents a feasible solution for the KSP if all free elements are setting equal to "zero". Let consider that, within each group, items are ranged in a decreasing order of the profit per weight. Then, the greedy algorithm, noted HEUR, can be described as follows:

**Input:** An instance of KSP.
**Output:** An approximate solution with its set of capacities.

***Starting.***
a) Set the initial capacity to zero, i.e. $SumCap = 0$; (cumulate total capacity)
b) Set $min = 1$, where $min$ denotes the index of the class realizing the minimum (sub)solution;
c) For each $i \in \{1, ..., m\}$, set $j_i = 1,\ P_i = 0$ and $W_i = 0$, where $P_i$ (resp. $W_i$) is the cumulate profit (resp. weight) of items picked in the $i$-th class.

***Iteration.***
1) If $SumCap + w_{j_{min}} \leq c$ then set $SumCap = SumCap + w_{j_{min}}$;
2) Set $j_{min} = j_{min} + 1$;
3) Let $min$ be an index realizing $\min_{1 \leq i \leq m} \{P_i\}$;
4) Repeat steps 1-3 till $j_{min} > |J_{min}|$.

Fig. 1. HEUR: an initial set of capacities-base solution for the KSP.

Note that each class $J_i$ contains a critical element, noted $r_{J_i}$, with a particular knapsack capacity $\bar{c}_{J_i}$ and so, the above procedure (HEUR) can terminate with a non-null residual capacity, i.e., $\sum_{i=1}^{m} \bar{c}_i \neq c$. Of course, in this case we can observe that by sharing the residual capacity ($c - \sum_{i=1}^{m} \bar{c}_i > 0$) over the classes (except for the class realizing the best solution), the provided solution remains feasible for the KSP.

### C. The main steps of the tree search algorithm

In this section, we describe the main steps of the tree search exact algorithm. In fact, the key of the algorithm is based on the result of the decomposition principle (Theorem 1 of Hifi and Sadfi [3]) applied for providing a nonnegative vector $\bar{c}$ = ($\bar{c}_1$, . . . , $\bar{c}_m$), where $\sum_{i=1}^{m} \bar{c}$'= $c$. We can remark that $\bar{c}$ is associated to an optimal solution (as shown in Hifi and Sadfi [3]) and so, the proposed algorithm applies the following two phases:

*Phase 1:* Decompose the KSP into a series of knapsack problems (see Section III-A) and apply HEUR in order to reach a starting set of capacities (see Section III-B).

*Phase 2:* At each step of the current phase, apply the dichotomous search procedure as detailed below.

We note that the second phase above is applied till a minimal solution value is attained; that is based on the following *optimality condition*.

**Theorem** 3.1: (see [5]) Let $\bar{c} = (\bar{c}_1, \ldots, \bar{c}_m)$ be a nonnegative vector with $\sum_{i=1}^{m} \bar{c}_i = c$, and $\ell$ be an index verifying the following equality:

$$VO(SK_{J_\ell}^{\bar{c}_\ell}) = VO(KSP) = \min\left\{VO(SK_{J_1}^{\bar{c}_1}), \ldots, VO(SK_{J_m}^{\bar{c}_m})\right\} \quad (1)$$

If $\forall i \neq \ell,\ i \in \{1, \ldots, m\}$, then

$$VO(SK_{J_\ell}^{\bar{c}_\ell}) \geq VO(SK_{J_\ell}^{(\bar{c}_i)-\alpha}), \quad (2)$$

where $\alpha$ denotes a bounded nonnegative integer such that $\alpha \leq \bar{c}_i \leq c$, then the vector $\bar{c}$ characterizes an optimal solution for KSP with value $VO(SK_{J_\ell}^{\bar{c}_\ell})$.
**Proof.** (see Hifi *et al.* [5]).

**Input:** an instance of the KSP.
**Output:** an optimal solution $Opt(KSP)$ of value $VO(KSP)$.

*Phase 1.* Initialization step
- Index the set of objets from 1 to $m$, i.e., $J_1, J_2, \ldots, J_m$;
- Index the set of elements of each class $J_i$ from 1 to $|J_i|$;

1) Let $\bar{c}'_i$, for $i = 1, \ldots, m$, be the components of the capacity vector $\bar{c}'$ provided by applying HEUR.
2) Apply a *branch and bound exact procedure* for solving the problems $SK_{J_i}^{\bar{c}'_i},\ i = 1, \ldots, m$, and denotes their solution values by $VO(SK_{J_i}^{\bar{c}'_i})$.
   - Let consider that $Opt(KSP)$ is the solution realizing the smallest value.
3) Set $\bar{c}_\ell \leftarrow \bar{c}'_\ell$, where $\ell$ denotes the index of the class realizing the smallest value $VO(SK_{J_i}^{\bar{c}'_i})$, for $i = 1, \ldots, m$.
4) Initialize the rest of the vector $\bar{c}$ using the `IntReduction()` procedure; that are, $\bar{c}_i \leftarrow \bar{c}''_i,\ i \neq \ell,\ i = 1, \ldots, m$, such that $\bar{c}''_i = \arg\min_{0 \leq \bar{c}''_i \leq \bar{c}'_i} \left\{VO(SK_{J_i}^{\bar{c}''_i}) > VO(SK_{J_\ell}^{\bar{c}_\ell})\right\}$.

Evaluating the initial $Gap_{\bar{c}}$ :
Compute the starting $Gap_{\bar{c}}$.

*Phase 2.* Iterative step
`While` ($Gap_{\bar{c}} > 0$) `Do`
a) Increment the capacity of the worst-class with the value $Gap_{\bar{c}}$; that is $\bar{c}_\ell \leftarrow \bar{c}_\ell + Gap_{\bar{c}}$;
b) Solve the current knapsack problem: $VO(SK_{J_\ell}^{\bar{c}_\ell}) \longleftarrow$ B&B$(\ell, J_\ell, \bar{c}_\ell)$;
c) Let $\ell$ be the class realizing the smallest solution value.
d) Use the following IRP for restarting the remaining capacity:
Set $\bar{c}_i \leftarrow \bar{c}''_i,\ i \neq \ell,\ i = 1, \ldots, m$, such that $\bar{c}''_i = \arg\min_{0 \leq \bar{c}''_i \leq \bar{c}'_i} \left\{VO(SK_{J_i}^{\bar{c}''_i}) > VO(SK_{J_\ell}^{\bar{c}_\ell})\right\}$.
e) Evaluating thecurrent value of $Gap_{\bar{c}}$ :
Compute the current value of $Gap_{\bar{c}}$.

`EndDo`
Exit with $Opt(KSP)$ realizing the value $VO(KSP) = \min_{1 \leq i \leq m} \left\{VO(SK_{J_i}^{\bar{c}_i})\right\}$.

Fig. 2. The exact algorithm for the KSP (ALGO1).

At each step of the algorithm we apply a Branch and Bound procedure following the main principle of Sahni [6], denoted $B\&B()$, to resolve each auxiliary problem $SK_{J_i}^{\bar{c}_i},\ i = 1, \ldots, m$. this last procedure is associated to a reduction interval procedure, denoted IntReduction(), used to compute the values of $\bar{c}_i,\ i = 1, \ldots, m$, at each step of the resolution.

**Theorem** 3.2: The algorithm stops after a finite number of iterations. Then, the provided solution represents an optimal solution to KSP.

## IV. Experimental part

In this section we evaluate the performance of the tree search exact algorithm (denoted ALGO1) on a series of instances taken from Hifi and Sadfi [3]. The algorithm is tested on two set of problem instances with different densities and sizes. ([1]). The first set contains the "strongly correlated" instances, and the second set is composed of the "uncorrelated" ones. The optimal solutions of these instances are known (see Hifi et Sadfi [3]) and in order to evaluate the behavior of ALGO1, we then compare its average runtime to that of the algorithm proposed in Hifi *et al.* [5], and noted ALGO. Both algorithms were coded in C++ and tested on an UltraSparc-II (450Mhz and with 2Gb of RAM).

First, as the behavior of both algorithm was equivalent on the small-sized instances, we then decided to make a comparative study on both medium and large sized instances.

TABLE I
PERFORMANCE OF ALGO1 VS ALGO ON THE "STRONGLY CORRELATED" INSTANCES

| Groupe | $T_{\text{ALGO1}}$ | $T_{\text{ALGO}}$ | Acc |
|---|---|---|---|
| D02.C | 2.00 | 127.50 | 63.75 |
| E02.C | 3.50 | 229.25 | 65.50 |
| F02.C | 20.00 | 915.50 | 45.78 |
| D05.C | 1.50 | 10.75 | 7.17 |
| E05.C | 2.00 | 25.75 | 12.88 |
| F05.C | 9.00 | 371.25 | 41.25 |
| D10.C | 1.00 | 5.00 | 5.00 |
| E10.C | 2.00 | 10.25 | 5.13 |
| F10.C | 5.00 | 54.25 | 10.85 |
| Average | 5.11 | 194.39 | 28.59 |

Second, Table I summarizes the behavior of both exact algorithms on the first set of problem instances. Column indicates the instance's name: we considered the set of instances $\mathtt{C}m\mathtt{C}, \ldots, \mathtt{F}m\mathtt{C}$ which correspond to the "strongly correlated" with the number of classes $m$ varying in the integer interval $[2, \ldots, 10]$ and the number $n$ of items varying in the integer interval $[7500, 20000]$. Column 2 contains the runtime (measured in seconds) that needs ALGO1 for reaching the optimal solutions. Column 3 tallies the computational time that needs ALGO, the best exact algorithm of the literature, for providing the optimal solution. Column 4 displays the acceleration realized by the proposed tree search algorithm ALGO1 compared to ALGO computed as follows: $T_{\text{ALGO}}/T_{\text{ALGO1}}$. Finally, the last line of the table summarizes the average runtime of both algorithms and the average acceleration realized by ALGO1.

From Table I, we can observe that ALGO1 outperforms ALGO on all instances of the first set. Indeed, the analysis of the results of the table revels that the average acceleration is equal to 28.59, for all treated "strongly correlated" instances.

[1]The benchmark instances are available on http://www.laria.u-picardie.fr/hifi/OR-Benchmark/KSP/KSP.html

In fact, it realizes a minimum of acceleration of 5 (instance D10.C) and it is able to realize a significant acceleration (for example, 65.50 times for instance E02.C).

TABLE II
BEHAVIOR OF BOTH ALGO1 AND ALGO ON THE "UNCORRELATED" INSTANCES

| Group | $T_{\text{ALGO1}}$ | $T_{\text{ALGO}}$ | Acc |
|---|---|---|---|
| D02 | 2.00 | 122.25 | 61.13 |
| E02 | 4.00 | 223.00 | 55.75 |
| F02 | 13.00 | 892.00 | 68.62 |
| D05 | 1.00 | 10.00 | 10.00 |
| E05 | 2.00 | 30.75 | 15.38 |
| F05 | 8.00 | 358.75 | 44.84 |
| D10 | 1.00 | 4.75 | 4.75 |
| E10 | 2.00 | 8.25 | 4.13 |
| F10 | 4.00 | 62.5 | 15.63 |
| D20 | 1.00 | 2.25 | 2.25 |
| E20 | 1.00 | 4.00 | 4.00 |
| F20 | 3.00 | 17.25 | 5.75 |
| D30 | 1.00 | 1.75 | 1.75 |
| E30 | 1.00 | 3.00 | 3.00 |
| F30 | 2.00 | 12.00 | 6.00 |
| D40 | 1.00 | 1.75 | 1.75 |
| E40 | 2.00 | 2.75 | 1.38 |
| F40 | 2.00 | 9.00 | 4.50 |
| D50 | 1.00 | 1.5 | 1.50 |
| E50 | 2.00 | 2.25 | 1.13 |
| F50 | 2.00 | 8.25 | 4.13 |
| Average | 2.67 | 84.67 | 15.11 |

Third and last, Table II shows the behavior of both algorithms ALGO1 and ALGO on the second set of instances containing large sized instances. Table II displays the same informations as for Table I , i.e., the runtime needed by both algorithms (ALGO and ALGO1, measured in seconds), the acceleration ($T_{\text{ALGO}}/T_{\text{ALGO1}}$) and a line containing the average rutime of each algorithm and the average acceleration realized by the proposed tree search algorithm.

From Table II, we can observe that the same phenomenon is realized. Indeed, we can remark that ALGO1 outperforms ALGO on all instances of the second set. In this case, ALGO1 realizes a minimum acceleration of 1.13 and a maximum one which attains the value of 68.62. Globally, the algorithm realizes an average acceleration of 15.11 which can be considered as a significant acceleration for an exact algorithm.

## V. CONCLUSION

We proposed an exact tree search algorithm for solving the knapsack sharing problem. First, we showed how the problem can be decomposed into a series of knapsack problems. Second, we presented the procedure providing the initial set of capacities-based solution; that is obtained by applying a greedy algorithm. Third, we showed how the branch and bound procedure can be applied for reaching an optimal solution for the problem. Finally, an experimental part has been presented in which we evaluated the performance of the proposed algorithm on a set of problem instances of the literature. On these problem instances, we proved experimentally the effectiveness of the proposed exact algorithm compared to the best exact algorithm of the literature.

## REFERENCES

[1] Brown JR. *The knapsack sharing*, Operations Research, 1979; 27:341-355.
[2] Brown JR. *Solving knapsack sharing with general tradeoff functions*, Mathematical Programming, 1991; 5:55-73.
[3] Hifi M., Sadfi S. *The knapsack sharing problem: an exact algorithm*, Journal of Combinatorial Optimization, 2002;6:35-54.
[4] Hifi M., Sadfi S., Sbihi A. *An efficient algorithm for the knapsack sharing problem*, Computational Optimization and Applications, 2002;23:27-45.
[5] Hifi M., MHalla H., Sadfi S. *An exact algorithm for the knapsack sharing problem*, to appear in Computers and Operations Research.
[6] Horowitz E., Sahni S. *Computing partitions with applications to the knapsack problem*, Journal of ACM, 1974;21:277-292.
[7] Kuno T., Konno H., Zemel E. *A linear-time algorithm for solving continuous maximum knapsack problems*, Operations Research Letters, 1991;10:23-26.
[8] Luss H. *Minmax resource allocation problems: optimization and parametric analysis*, European Journal of Operational Research, 1992;60:76-86.
[9] Martello S., Toth P (eds.). Knapsack problems: algorithms and computer implementation, John Wiley and Sons, 1990.
[10] Martello S, Toth P. *Upper bounds and algorithms for hard 0-1 knapsack problems*, Operations Research, 1997;45:768-778.
[11] Pang JS., Yu CS. *A min-max resource allocation problem with substitutions*, European Journal of Operational Research, 1989;41:218-223.
[12] Tang CS. *A max-min allocation problem: its solutions and applications*, Operations Research, 1988;36:359-367.
[13] Yamada T., Futakawa M. *Heuristic and reduction algorithms for the knapsack sharing problem*, Computers and Operations Research, 1997;24:961-967.
[14] Yamada T., Futakawa M., Kataoka S. *Some exact algorithms for the knapsack sharing problem*, European Journal of Operational Research, 1998;106:177-183.