Solving the 0-1 Multidimensional Knapsack Problem with Resolution Search

Sylvain Boussier, Michel Vasquez, Yannick Vimont, Said Hanafi, Philippe Michelon

Solving the 0–1 Multidimensional Knapsack Problem with Resolution Search

Sylvain Boussier[a], Michel Vasquez[a], Yannick Vimont[a], Saïd Hanafi[b] and Philippe Michelon[c]

[a]*LGI2P, Ecole des Mines d'Alès, Parc scientifique Geroges Besse, 30035 Nîmes, France,* {Sylvain.boussier,Michel.Vasquez,Yannick.Vimont}@ema.fr
[b]*LAMIH, Université de Valenciennes et du Hainaut-Cambrésis Le Mont Houy - BP 311, 59304 Valenciennes, France,* Said.Hanafi@univ-valenciennes.fr
[c]*LIA, Université d'Avignon et des Pays de Vaucluse, 339 chemin des Meinajaries, BP 1228, 84911 Avignon Cedex 9, France,* Philippe.Michelon@univ-avignon.fr

Keywords: 0–1 Multidimensional Knapsack Problem, Resolution Search, Branch & Bound

Abstract. We propose an exact method which combines the resolution search and branch & bound algorithms for solving the 0–1 Multidimensional Knapsack Problem. This algorithm is able to prove large–scale strong correlated instances. The optimal values of the 10 constraint, 500 variable instances of the OR-Library are exposed. These values were previously unknown.

1 Introduction

In this article, we present a new exact method which hybridizes the resolution search of Chvátal (1997) and a branch & bound algorithm inspired by a previous work from Vimont et al. (2008) for the 0–1 Multidimensional Knapsack Problem (01MKP). The 01MKP is a well–known opitmization problem which can be viewed as a resource allocation model and can be stated as follows:

$$(P) \quad \text{Maximize} \quad \sum_{j=1}^{n} c_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, ..., m \tag{2}$$

$$x_j \in \{0, 1\} \qquad j = 1, ..., n \tag{3}$$

where n is the number of items, m is the number of knapsack constraints with capacities b_i ($i = 1, 2, ..., m$), $c \in \mathbb{N}^n$, $A \in \mathbb{N}^{m \times n}$ and $b \in \mathbb{N}^m$. Each item j ($j = 1, 2, ..., n$) yields c_j units of profit and consumes a given amount of resource a_{ij} for each knapsack i.

The proposed approach is centered on two main 01MKP results: (i) the consideration of a reduced costs constraint based on the reduced costs at the optimality of the problem's LP–relaxation (see Balas and Martin (1980); Oliva et al. (2001)) and (ii) the decomposition of the search space in several hyperplane where the number of items to choose is fixed at a given integer value (see Vasquez and Hao (2001); Vimont et al. (2008)). Our algorithm is self–sufficient and does not require any lower bound as starting value. We show that the structure of resolution search enables to explore partially and iteratively different subproblems (*hyperplanes with a fixed number of items*) while keeping completeness. This way of exploration enhances the diversification of the search and permits to improve the lower bound rapidly. For each hyperplane, this lower bound, associated with the upper bound (*tighter than the classical one given by the*

LP–relaxation), enforces the strength of the so–called *reduced costs constraint* widely used in the algorithm. Roughly speaking, if we consider that the variables are sorted in decreasing order of their reduced costs (*basic variables at the bottom of the list*), the enumeration of the first variables is carried out by resolution search where the enumeration of the remaining variables is tackled by our specific branch & bound. Our approach proved all the optimality of all the 10 constraint, 500 variable instances. These optimal solutions were previously unknown.

2 Resolution search

Resolution search was proposed by Chvátal (1997) as an alternative to branch & bound for mixed 0–1 linear programming problems. This approach, based on an original exploration of the search space, uses the information brought by the fails that have occurred during the search to progressively shrink the search tree. Each time a terminal node is reached, a minimal partial instantiation of the variables responsible for the fail is identified. This partial instantiation, which corresponds also to a terminal node, is then recorded in a specific way in order to discard the corresponding subtree from the search space and to provide the next node in the exploration. The specificity of the recording mechanism allows the algorithm to preserve space memory while keeping completeness.

Globally, resolution search is composed of two main elements: (i) a set of partial instantiations (denoted path–like family $\mathscr{F}$) corresponding to fails encountered during the search and recorded as boolean clauses and (ii) an function called `obstacle` which uses the information brought by the family $\mathscr{F}$ to explore promising parts of the search space. From a partial instantiation $u(\mathscr{F})$ derived from $\mathscr{F}$, which is not an already explored instantiation, `obstacle` performs two different phases:

1. *The waxing phase* which replaces step by step the free components $u(\mathscr{F})_j = *$ by 0 or 1 until a terminal node (denoted u^*) is reached (a terminal node is a partial instantiation of the variables which does not lead to any improvement of the best known lower bound).

2. *The waning phase* which tries to find a minimal element S of $\{0,1,*\}^n$ such that $S \sqsubseteq u^*$ and S is a terminal node (formally, let u and v be two vectors of $\{0,1,*\}^n$, we call u an *extension* of v and we write $u \sqsubseteq v$, if $v_j = u_j$ whenever $u_j \neq *$).

Once the minimal element S is identified, it is recorded in a specific way as a clause in $\mathscr{F}$. The structure of $\mathscr{F}$ enables both to guide the search toward prosimising parts of the search space and to guaranteed the completeness using the resolution / refutation principle. Indeed, if the minimal element S correspond to the instantiation $(*,*,...,*)$ then the best known solution is an optimal soltion of P. The algorithm 1. represents the function `Resolution_Search` which takes as parameter a lower bound (LB) of the problem.

3 General approach

The 01MKP is tackled by decomposing the problem in several subproblems where the number of items to choose is fixed at a given integer value. Considering that LB is a lower bound of the problem, we can define $X = \{x \mid A \cdot x \leq b, c \cdot x \geq LB + 1, x \in \{0,1\}^n\}$ as the set of feasible solutions strictly better than LB and consider the following problems:

$$P^+ : \text{ maximize } \{1 \cdot x \mid x \in \bar{X}\} \quad \text{and} \quad P^- : \text{ minimize } \{1 \cdot x \mid x \in \bar{X}\}$$

where 1 is the vector of dimension n with all its components equal to one and $\bar{X} = \{x \mid A \cdot x \leq b, c \cdot x \geq LB + 1, x \in [0,1]^n\}$. In what follows, we note $v(P)$ the optimal value of a given

Figure 1: Resolution Search Algorithm

```
Resolution_Search(LB)
{
   F = ∅;
   u = (*, *, ..., *);
   while(1) {
      try = obstacle(u,LB,S);
      if(try > LB)  LB = try;
      add S to F and update F;
      u = u(F);
      if((*, *, ..., *) ∈ F)  Break;
   }
}
```

problem P. Let $k_{min} = \lceil v(P^-) \rceil$ and $k_{max} = \lfloor v(P^+) \rfloor$, then we have $v(P) = max\{v(P_k) \mid k_{min} \leq k \leq k_{max}\}$ where

$$P_k : \text{ maximize } \{cx \mid x \in X, 1 \cdot x = k\}.$$

Solving the 01MKP by tackling separately each of the subproblems P_k for $k = k_{min}, ..., k_{max}$ appeared to be an interesting approach (Vasquez and Hao (2001); Vasquez and Vimont (2005); Vimont et al. (2008)) particularly because the additional constraint $(1 \cdot x = k)$ provides tighter upper bounds than the classical LP–relaxation. This approach takes also into consideration a constraint based on the reduced costs of the non–basic variables. Let us consider the upper bound $UB = c \cdot \bar{x}$, where $\bar{x}$ is the optimal solution of the LP–relaxation. Let $\bar{c}$ be the vector of the reduced costs and N the indexes of the non–basic variables. If we know a lower bound $LB \in \mathbb{N}$ of the problem, then each better solution x must satisfy the following constraint:

$$\sum_{j \in N | \bar{x}_j = 0} |\bar{c}_j| x_j + \sum_{j \in N | \bar{x}_j = 1} |\bar{c}_j| (1 - x_j) \leq UB - LB \tag{4}$$

(the reader is referred to Balas and Martin (1980); Oliva et al. (2001) and Vimont et al. (2008) for more details on the subject). The use of the constraint (4), also called the *reduced costs constraint*, is twofold: (i) it enables us to identify the unfeasibility (with respect to the constraint $c \cdot x \geq LB + 1$) of some partial solutions and (ii) it allows us to fix those variables x_j with $|c_j| > UB - LB$ at their optimal value $\bar{x}_j$. Both the tight upper bound and the good lower bound enhance the efficiency of the reduced costs constraint.

To summarize, our approach consists of decomposing the search space in several hyperplanes then solving the LP-relaxation for each one in order to generate the reduced costs constraint. The exploration is then carried out partially and iteratively for each hyperplane with resolution search until all the search space is explored.

Indeed, thanks to the structure of resolution search, P can be solved by progressively exploring each of the hyperplanes $(1 \cdot x = k,\ k \in \mathbb{N})$. Let `CFamily[`$k$`]` be the family of clauses associated to resolution search for the subproblem P_k. At each step of the search, `CFamily[`k`]` provides all the information about the state of the search: the terminal nodes recorded at this stage and the next node to explore. It is possible to execute some iterations of resolution search at a given P_k, then continue to another $P_{k'}$ and go on with the subproblem P_k again without any loss of information. The algorithm 2 detail the `Iterative_RS` algorithm which corresponds to resolution search limited to a given number of iterations and the algorithm 3 shows

the hyperplane exploration using `Iterative_RS`. This way of exploration enforces the diversification and the convergence of the search. Note that, in algorithm 3, `greedy` is a simple greedy function used for providing a first lower bound.

```
Iterative_RS(Nb_Iter, LB, F)
{
   iter = 0;
   while(iter < Nb_Iter) {
      try = obstacle( u(F),LB,S);
      if(try > LB)  LB = try;
      add S to F and update F;
      if((*,*,...,*) ∈ F)  Break;
      iter++;
   }
}
```

Figure 2: Iterative resolution search

```
LB = greedy();
Compute the bounds k_min and k_max;
Set K = {k_min, ..., k_max;}
for(k = k_min; k ≤ k_max; k++)
   CFamily[k] = ∅
While (K ≠ ∅) do {
   Choose k ∈ K;
   Choose Nb_Iter_k ≥ 1;
   Iterative_RS(Nb_iter_k,LB,CFamily[k]);
   if((*,*,...,*) ∈ CFamily[k])  K = K − {k};
}
```

Figure 3: Hyperplanes exploration

4 Resolution search and branch & bound combination

In this section, we detail the exploration function `obstacle` embedding the branch & bound algorithm.

Starting with the node $u(\mathscr{F})$ given by the path–like family $\mathscr{F}$, `obstacle` replaces step–by–step the components $*$ of $u(\mathscr{F})$ by 0 or 1 which constructs the node u^+. If u^+ is a terminal node, the function provides a minimal clause S such that $S \sqsubseteq u^*$ and S is a terminal node. Our implementation of `obstacle` is based on the reduced–costs constraint (4) presented in section 3. At the beginning of the procedure, the LP–relaxation of the problem is solved for each available hyperplane in order to give us the information needed for the reduced costs constraint. Then `obstacle` proceeds to the following steps:

- The first step, called *consistency phase*, consists of checking the feasibility of $u(\mathscr{F})$. Initially, we define a value $gap = UB - LB$. If a constraint is violated, the descent phase stops and the corresponding partial instantiation is recorded as the clause S in $\mathscr{F}$. At the same time, the reduced costs constraint (4) is checked and the gap value is updated: for each non–basic variable set at the opposite of its optimal value ($1 - \bar{x}_j$), its reduced costs ($\bar{c}_j$) is subtracted from the gap. If it happens that $gap < 0$ then the current partial solution is a terminal node. In this case, S is only composed of the variables set at the opposite of their optimal value in $u(\mathscr{F})$.

- If $u(\mathscr{F})$ is feasible, we go to the next step which we call *implicit waning phase*. This phase consists of branching on all the remaining free variables with a reduced cost greater than gap. Those variables must be fixed at their optimal value for satisfying the reduced costs constraint (4). The branching decisions taken in this phase are just a consequence of the instantiated variables in $u(\mathscr{F})$. Consequently they can be removed from S.

- Then the algorithm starts the so–called *waxing phase* which consists of assigning values to free variables. The chosen strategy is to select the free variable with the greater absolute reduced cost value and to assign its optimal value $\bar{x}_j$ to it. Obviously, each time a branching is done, the feasibility of the current partial solution is verified and in case of fail, the waxing phase stops and the corresponding clause S is added to $\mathscr{F}$. Note that the variables set in the implicit waning phase are still not taken into account in S.

- Once the number of remaining free variables is less than or equal to a given number `spb_size`, the waxing phase stops and the corresponding subproblem is solved with a branch & bound algorithm. This subproblem includes the free variables with the lowest reduced cost and the basic variables. Obviously, since the branch & bound algorithm explores the whole subtree corresponding to these variables, the clause S does not contain any branching choices made during this phase. Only the branching decisions taken during the consistency phase *and/or* the waxing phase are considered.

- The algorithm used to enumerate the variables with the lowest reduced cost, and the basic variables, is widely inspired by a previous one published by Vimont et al. (2008). As it is represented in figure 4 (which summarizes the exploration process), this algorithm embeds a specific Depth First Search (dfs) procedure for solving small subproblems with the last 20 variables.

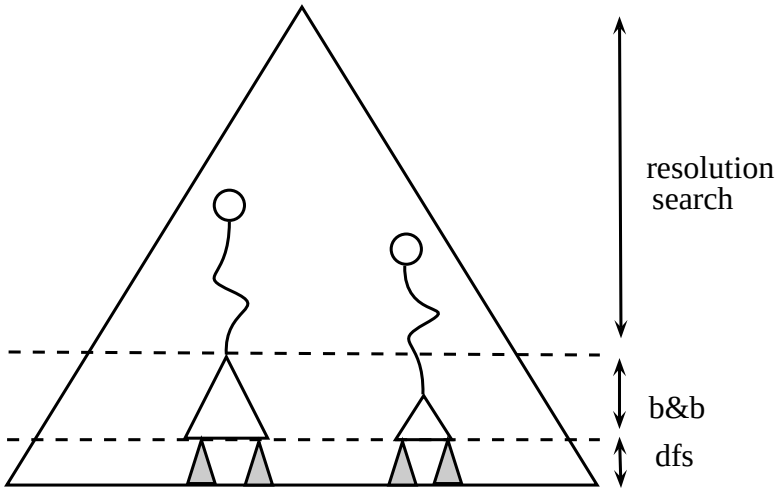

Figure 4: Global view of the exploration process of an hyperplane

5 Computational results

Our algorithm has been experimented on the well known OR-Library set of 01MKP instances proposed by Beasley (1990). Each instance of this set is denoted cb$m.n_r$ where m is the number of constraints, n the number of variables and r the instance's number. The 01MKP OR-Library benchmark is composed of 5, 10 and 30 constraint instances and 100, 250 and 500 variables instances. There are 9 set of 30 instances with each $n \times m$ combination.

We obtained better proof time than the exact published approaches of James and Nakagawa (2005), Vimont et al. (2008) and the commercial software CPLEX 9.2, for the 10 constraint, 250 variable instances and the 5 constraint, 500 variable instances. We observed also that for the 5 constraint, 500 variable instances, the times required for obtaining the best solutions are clearly better than the ones provided by the best known heuristics on these instances (Vasquez and Vimont (2005) and Wilbaut and Hanafi (2008)). Moreover, our algorithm proved the optimality of all the cb10.500 instances. The corresponding optimal values, which were previously unknown, are exposed in table 1. The column z^{opt} is the optimal value of the instance, *opt. (h)* is the time in hours required for obtaining the optimal value and *proof (h)* is the time in hours for proving the optimality of the value z^{opt}. The column $z^{opt} - \underline{z}$ corresponds to the gap between the optimal value and the previously best known solution: *(vv)* indicates that $\underline{z}$ was first found by Vasquez and Vimont (2005) and *(wh)* indicates that $\underline{z}$ was first found by Wilbaut and Hanafi (2008).

Instance	z^{opt}	opt. (h)	proof (h)	$z^{opt} - \underline{z}$	Instance	z^{opt}	opt. (s)	proof (h)	$z^{opt} - \underline{z}$
cb10.500_0	117821	24,5	567,2	**+10**(vv)	cb10.500_15	215086	0	43,9	0
cb10.500_1	119249	68,4	272,9	**+17**(vv)	cb10.500_16	217940	13,4	36,1	0
cb10.500_2	119215	18,6	768,3	0	cb10.500_17	219990	150,8	348,8	0
cb10.500_3	118829	47,4	89,6	**+4**l(wh)	cb10.500_18	214382	12,7	57,8	**+7**(vv)
cb10.500_4	116530	86,1	2530,3	**+16**(wh)	cb10.500_19	220899	0,2	21,3	0
cb10.500_5	119504	2,3	188	0	cb10.500_20	304387	6,6	8,2	0
cb10.500_6	119827	2,7	128	0	cb10.500_21	302379	0	8,4	0
cb10.500_7	118344	161,7	179,6	**+11**(wh)	cb10.500_22	302417	67,2	105,5	**+1**(vv)
cb10.500_8	117815	86,3	219,9	0	cb10.500_23	300784	0,9	3,8	0
cb10.500_9	119251	3,1	354,9	0	cb10.500_24	304374	0,1	16,8	0
cb10.500_10	217377	0	515,8	0	cb10.500_25	301836	29,7	30,9	0
cb10.500_11	219077	0,5	437,6	0	cb10.500_26	304952	0	18,5	0
cb10.500_12	217847	0	5,5	0	cb10.500_27	296478	1,1	9,3	0
cb10.500_13	216868	0	104,4	0	cb10.500_28	301359	8,1	39,1	0
cb10.500_14	213873	59,4	1382,1	**+14**(vv)	cb10.500_29	307089	1,2	4,4	0

Table 1: Results obtained on the 10 constraint, 500 variable instance of the `OR-Library`

6 Conclusion

Although our implementation of resolution search is quite far from the original one proposed by Chvátal, especially because we have exploited specific structures of the 01MKP (hybridization with branch & bound, $1.x = k$ hyperplane decomposition, reduced cost constraint to generate implicitly the partial instantiations responsible for the fails, etc.), we showed that resolution search is a promising framework for designing efficient algorithms. Since the proof times are long for the 10 constraint, 500 variable instances, we plan to improve again our algorithm in order to accelerate the resolution process in the hope of maybe closing the OR-Library 01MKP benchmark by solving the 60 last 30 constraint 250/500 variable instances.

References

Balas, E. and Martin, C. H.: 1980, Pivot and complement a heuristic for zero-one programming, *Management Science* **26**(1), 86–96.

Beasley, J.: 1990, Or-library: Distributing test problems by electronic mail., *J. Operational Research Society* **41**, 1069–1072.

Chvátal, V.: 1997, Resolution search, *Discrete Applied Mathematics* **73**, 81–99.

James, R. and Nakagawa, Y.: 2005, Enumeration methods for repeatedly solving multidimensional knapsack sub-problems, *Technical Report 10*, The Institute of Electronics, Information and Communication Engineers.

Oliva, C., Michelon, P. and Artigues, C.: 2001, Constraint and linear programming : Using reduced costs for solving the zero/one multiple knapsack problem, *CP 01, Proceedings of the workshop on Cooperative Solvers in Constraint Programming (CoSolv 01)* pp. 87,98.

Vasquez, M. and Hao, J.: 2001, An hybrid approach for the 0-1 multidimensional knapsack problem., *Proc. 17th International Joint Conference on Artificial Intelligence, IJCAI-01, Seattle, WA* .

Vasquez, M. and Vimont, Y.: 2005, Improved results on the 0-1 multidimensional knapsack problem., *European Journal of Operational Research* (165), 70–81.

Vimont, Y., Boussier, S. and Vasquez, M.: 2008, Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem, *Journal Of Combinatorial Optimization* **15**(2), 165–178.
URL: *http://dx.doi.org/10.1007/s10878-007-9074-4*

Wilbaut, C. and Hanafi, S.: 2008, New convergent heuristics for 0-1 mixed integer programming, *European Journal of Operational Research* **doi:10.1016/j.ejor.2008.01.044.**