A Hybrid Algorithm for the Unbounded Knapsack Problem

Vincent Poirriez, Nicola Yanev, Rumen Andonov

A Hybrid Algorithm for the Unbounded Knapsack Problem

Vincent Poirriez [a] Nicola Yanev [b] Rumen Andonov [c,*]

[a] *LAMIH/ROI UMR CNRS 8530, University of Valenciennes, Le Mont Houy, 59313 Valenciennes Cedex 9, France*

[b] *Faculty of Mathematics and Informatics, University of Sofia, 1164 Sofia, 5 James Bourchier Blvd., Bulgaria*

[c] *INRIA Rennes-Bretagne Atlantique and University of Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France*

Abstract

This paper presents a new approach for exactly solving the Unbounded Knapsack Problem (UKP) and proposes a new bound that was proved to dominate the previous bounds on a special class of UKP instances. Integrating bounds within the framework of sparse dynamic programming led to the creation of an efficient and robust hybrid algorithm, called EDUK2. This algorithm takes advantage of the majority of the known properties of UKP, particularly the diverse dominance relations and the important periodicity property. Extensive computational results show that, in all but a very few cases, EDUK2 significantly outperforms both MTU2 and EDUK, the currently available UKP solvers, as well the well-known general purpose mathematical programming optimizer CPLEX of ILOG. These experimental results demonstrate that the class of hard UKP instances needs to be redefined, and the authors offer their insights into the creation of such instances.

Key words: Combinatorial Optimization, Integer Programming, Knapsack problem, Branch and Bound, Dynamic programming, Algorithm Engineering

1 Introduction

The knapsack problem is one of the most popular combinatorial optimization problems. Its unbounded version, UKP (also called the integer knapsack), is formulated as follows: there is a knapsack of a *capacity* $c > 0$ and n types of items. Each

* Corresponding author.

Email addresses: vincent.poirriez@univ-valenciennes.fr (Vincent Poirriez), choby@math.bas.bg (Nicola Yanev), randonov@irisa.fr (Rumen Andonov).

item of type $i \in I = \{1, 2, \ldots n\}$ has a *profit*, $p_i > 0$, and a *weight*, $w_i > 0$. Set $N = \{(p_i, w_i), i \in I\}$ and let $\mathbf{w}, \mathbf{p}$ denote vectors of size n. The problem, $\mathbf{UKP}^{c}_{\mathbf{w},\mathbf{p}}$, is to fill the knapsack in an optimal way, which is done by solving

$$f(N, c) \equiv f(\mathbf{w}, \mathbf{p}, c) = \max\left\{\mathbf{px} \text{ subject to } \mathbf{wx} \leq c, \mathbf{x} \in Z^n_+\right\} \tag{1}$$

where Z^n_+ is the set of nonnegative integral n-dimensional vectors.

Many of this problem's properties have been discovered over the last three decades: [1,4,6,11,10,14], but no existing solver has yet been developed that benefits from all of them. A detailed and comprehensive state-of-the art discussion the interested reader can find in the recent monograph [12].

In this paper we introduce *a new upper bound* and determine a UKP family for which this bound is the tightest one known. We also design a new algorithm that combines dynamic programming and branch-and-bound methods to solve UKP. To the best of our knowledge this is the first time that such an approach has been used for UKP. Extensive computational experiments demonstrate the effectiveness of embedding a branch-and-bound algorithm into a dynamic programming framework. These results also shed light on the case of really hard UKP instances.

A hybrid algorithm, combining dynamic programming and branch-and-bound approaches has been proposed in [8] for the 0/1 knapsack problem, and in [9] for the case of the subset-sum problem. The adjective "hybrid" was also used for knapsack problem algorithms in [13] (0/1 knapsack problem) and [3] (0/1 multidimensional knapsack problem), but this is another kind of hybridization.

The paper is organized as follows. Section 2 briefly summarizes the basic properties of the problem. Section 3 presents a new upper bound and the associated class of instances where it is stronger than the previously known bounds. Section 4 is dedicated to the description of `EDUK2`, a new algorithm that takes advantage of all known dominance relations and successfully combines them with a variety of bounds [1] . In Section 5 this algorithm is compared with other available solvers. In Section 6 we conclude.

2 A summary of known dominance relations and bounds

The dominance relations between items and bounds allow the size of the search space to be significantly reduced. All the dominance relations, enumerated below,

[1] `EDUK2` is free open-source software available at:
`http://download.gna.org/pyasukp/` where it is denoted by PYAsUKP.

could be derived by the following inequalities:

$$\sum_{j\in J} w_j x_j \le \alpha w_i, \text{ and } \sum_{j\in J} p_j x_j \ge \alpha p_i \text{ for some } \mathbf{x} \in Z_+^n \tag{2}$$

where $\alpha \in Z_+$, $J \subseteq I$ and $i \notin J$.

(1) *Dominances*

(a) *Collective Dominance* [1,17]. The i-th item is **collectively dominated** by J, written as $i \ll J$ iff (2) hold when $\alpha = 1$. The verification of this dominance is computationally hard, so it can be used in a dynamic programming approach only. To the best of our knowledge EDUK (Efficient Dynamic programming for UKP) [1] is the only one that makes practical use of this property.

(b) *Threshold Dominance* [1]. The i-th item is **threshold dominated** by J, written as $i \lessdot J$ iff (2) hold when $\alpha \ge 1$. This is an obvious generalization of the previous dominance by using instead of single item i a compound one, say α times item i. The smallest such α defines the **threshold** of the item i, written t_i, as $t_i = (\alpha - 1)w_i$.

The lightest item of those with the greatest profit/weight ratio is called *best item*, written as b. One can trivially show that $t_i \le w_b w_i$ or even sharper inequality $t_i \le lcm(w_b, w_i)$ where $lcm(w_b, w_i)$ is the least common multiple of w_i and w_b.

(c) *Multiple Dominance* [10]. Item i is **multiply dominated** by j, written as $i \ll_m j$, iff for $J = \{j\}, \alpha = 1$, $x_j = \lfloor \frac{w_i}{w_j} \rfloor$ the relations (2) hold.

This dominance could be efficiently used in a preprocessing because it can be detected relatively easily.

(d) *Modular Dominance* [17]. Item i is **modularly dominated** by j, written as $i \ll_{\equiv} j$ iff for $J = \{b, j\}, \alpha = 1, w_j = w_i + tw_b, t \le 0$, $x_b = -t, x_j = 1$ the inequalities (2) hold.

(2) *Bounds*

U_3 **[10]** : It is assumed that the first three items are of the largest profit/weight ratio. Let us set

$$\bar{c} = c \bmod w_1;\ c' = \bar{c} \bmod w_2;\ z' = \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + \left\lfloor \frac{\bar{c}}{w_2} \right\rfloor p_2;$$

$$U^0 = z' + \left\lfloor \frac{c' p_3}{w_3} \right\rfloor;$$

$$\bar{U}^1 = z' + \left\lfloor \left(c' + \left\lceil \frac{w_2 - c'}{w_1} \right\rceil w_1 \right) \frac{p_2}{w_2} - \left\lceil \frac{w_2 - c'}{w_1} \right\rceil p_1 \right\rfloor.$$

The following bound holds

$$U_3 = \max\{U^0, \bar{U}^1\}. \tag{3}$$

U_s **[4]** : $U_s = c + \left\lfloor \frac{c}{w_1} \right\rfloor \alpha$, where item 1 is supposed to be the lightest one. It could be easily shown that this bound is valid (but could be very weak) for arbitrary UKP with α such that $p_i \leq w_i + \alpha$. It is proved in [4] that this bound is stronger than U_3 for the class of strongly correlated UKP (**SC-UKP**) defined as $p_i = w_i + \alpha$ where $\alpha > 0$. The case $\alpha = 0$ corresponds to the so called Subset Sum Problem (**SS-UKP**) where $p_i = w_i$.

U_v **[15]** [2]: $U_v = c + \max\left\{\frac{(p_i - w_i)}{\lfloor \frac{w_i}{w_1} \rfloor}, i \in I\right\} \left\lfloor \frac{c}{w_1} \right\rfloor$. Here again item 1 is supposed to be the lightest one. This bound is stronger than U_3 for a special class of UKP (namely **SAW-UKP** see Definition 1 below).

3 A new general upper bound for UKP

In the following paragraphs, we introduce a new upper bound for the UKP and show that it improves U_v and is not comparable to U_3 in the general case. For the special UKP family, the **SAW-UKP**, which includes the **SC-UKP** class (with $\alpha \geq 0$), this new bound is tighter than the previously known bounds.

Without losing generality it is assumed in this section that: *1 is the lightest item* within the set of items with $(p_i - w_i) > 0$ (i.e. $\forall i > 1,\ w_1 \leq w_i$ or $p_i \leq w_i$) and $p_1 > w_1$. (If all $p_i - w_i \leq 0$ then assume 1 is the item with the best ratio and by changing $\mathbf{p}$ to $\psi\mathbf{p}, \psi > \frac{w_1}{p_1}$, we will achieve the goal. If such an equivalent transformation is done, the bound should be divided by ψ). It is also assumed that no item is multiply dominated. Let us define the following terms:

for k fixed, for all $i \neq k, q_k^i = \frac{p_i - p_k \left\lfloor \frac{w_i}{w_k} \right\rfloor}{w_i - w_k \left\lfloor \frac{w_i}{w_k} \right\rfloor}$, $q_k^* = \max_{i \neq k} \left\{q_k^i\right\}$,

$\tau_1^* = \min\{1, q_1^*\}$, $\beta_1(\tau) = \max_{i \in I} \left\{\frac{p_i - \tau w_i}{\lfloor \frac{w_i}{w_1} \rfloor}\right\}$, $\beta_1^* = \beta(\tau_1^*)$.

Theorem 1 *[U_{τ^*}]* *for all UKP$^c_{\mathbf{w},\mathbf{p}}$, $f(\mathbf{w}, \mathbf{p}, c) \leq U_{\tau^*} = \tau_1^* c + \beta_1^* \left\lfloor \frac{c}{w_1} \right\rfloor \leq U_v$*

Proof: First, for any fixed $\tau \geq 0$,

$$\begin{aligned} \max\{\mathbf{px}, \mathbf{wx} \leq c, \mathbf{x} \in Z_+^n\} &= \max\{\tau\mathbf{wx} + (\mathbf{p} - \tau\mathbf{w})\mathbf{x}, \mathbf{wx} \leq c, \mathbf{x} \in Z_+^n\} \\ &\leq \tau c + \max\{(\mathbf{p} - \tau\mathbf{w})\mathbf{x}, \mathbf{wx} \leq c, \mathbf{x} \in Z_+^n\} \quad (4) \end{aligned}$$

Case $\tau_1^* = q_1^* \leq \tau \leq 1$**:** in this case, $q_1^* = \max_{i \neq 1} \left\{ \frac{p_i - p_1 \left\lfloor \frac{w_i}{w_1} \right\rfloor}{w_i - w_1 \left\lfloor \frac{w_i}{w_1} \right\rfloor} \right\} \leq \tau$

[2] First presented in a research report [15], this bound is also used in [12].

and therefore

$$\text{for all } i, p_i - \tau w_i \leq \left\lfloor \frac{w_i}{w_1} \right\rfloor (p_1 - \tau w_1). \tag{5}$$

Relation (5) means that in $\text{UKP}^{c}_{\mathbf{w},(\mathbf{p}-\tau\mathbf{w})}$ all items i are multiply dominated by the item 1, and also that $\beta_1(\tau) = p_1 - \tau w_1$. Thus, $\max\{(\mathbf{p}-\tau\mathbf{w})\mathbf{x}, \mathbf{w}\mathbf{x} \leq c, \mathbf{x} \in Z^n_+\} = \beta_1(\tau) \left\lfloor \frac{c}{w_1} \right\rfloor$.

The function $u_1(\tau) = \tau c + (p_1 - \tau w_1) \left\lfloor \frac{c}{w_1} \right\rfloor$ is an increasing function, and its minimum is reached for $\tau = \tau_1^*$. This proves both inequalities of the theorem as $U_v = u_1(1)$ and $U_{\tau^*} = u_1(\tau_1^*)$.

Case $q_1^* > 1 = \tau_1^*$**:** in this case,

$$\Sigma_{i=1}^n (p_i - w_i)x_i \leq \beta_1^* \Sigma_{i=1}^n \left\lfloor \frac{w_i}{w_1} \right\rfloor x_i \leq \beta_1^* \left\lfloor \Sigma_{i=1}^n \frac{w_i x_i}{w_1} \right\rfloor \leq \beta_1^* \left\lfloor \frac{c}{w_1} \right\rfloor$$

and $U_{\tau^*} = U_v = c + \beta_1^* \left\lfloor \frac{c}{w_1} \right\rfloor$. ∎

Let us set $uu(\tau) = \tau c + f(\mathbf{w}, \mathbf{p} - \tau\mathbf{w}, c)$ (defined for $\tau \geq 0$). We have $f(\mathbf{w}, \mathbf{p}, c) = uu(0)$. Furthermore, it follows from (4) that $f(\mathbf{w}, \mathbf{p}, c)$ is upper-bounded by $uu(\tau)$, which is a nondecreasing piece-wise linear convex function. One known point on its graphics is at $\tau_b = \frac{p_b}{w_b}$. A better bound is provided by the points $(\tau, uu(\tau))$, $\tau < \tau_b$. In the first case of the proof, $q_1^* \leq 1$, such a point is given by $(q_1^*, uu(q_1^*))$. When $q_1^* > 1$ (far from the target $\tau = 0$) we can overestimate $uu(\tau)$ in a point closer to 0, (say $\tau = 1$). Such an estimate is done in the case 2 from above, but it is quite rough (because of overestimating the $p - \tau w$ coefficients and in the rounding operation $\lfloor \sum \frac{w_i x_i}{w_1} \rfloor$ instead of $\sum \lfloor \frac{w_i x_i}{w_1} \rfloor$).

Another approach is demonstrated in the theorem below, with the main idea to "visualize" the graphics of $uu(\tau)$ from the left of the point $\frac{p_b}{w_b}$. This is done by changing the role of item 1 with item k, where k is such that $q_k^* \leq \tau \leq \frac{p_k}{w_k}$ is solvable. In the following theorem this is the case $k = b$.

Theorem 2 *The bound $U_b^* = q_b^* c + (p_b - q_b^* w_b) \lfloor \frac{c}{w_b} \rfloor$ is stronger than the (classical) upper bound $U = \frac{p_b c}{w_b}$, and it is strictly stronger when c is not a multiple of w_b and $q_b^* < \frac{p_b}{w_b}$.*

Proof: The idea of the proof is quite simple: $f(\mathbf{w}, \mathbf{p}, c) \leq \tau c + f(\mathbf{w}, \mathbf{p} - \tau\mathbf{w}, c)$ holds for arbitrary $\tau \geq 0$. When $\tau \geq q_b^*$, similarly to the first case of Theorem 1, we can show that the best item b multiply dominates all other items, thus giving the optimal $x_b = \lfloor \frac{c}{w_b} \rfloor$ solution to the knapsack $\text{UKP}^{c}_{\mathbf{w},(\mathbf{p}-\tau\mathbf{w})}$ with value $(p_b - \tau w_b)x_b$. It is easy to check that $q_b^* = \max_{i \neq b} \left\{ q_b^i \right\} \leq \frac{p_b}{w_b} \Leftrightarrow \frac{p_i}{w_i} \leq \frac{p_b}{w_b}$. Furthermore, $u_b(\tau) = \tau c + (p_b - \tau w_b) \lfloor \frac{c}{w_b} \rfloor$ is an increasing function, and gives a better upper bound than $U = u_b(\frac{p_b}{w_b})$ when $q_b^* \leq \tau \leq \frac{p_b}{w_b}$.

The second half of the theorem follows from the observation that $u_b(\tau)$ is strictly increasing when c is not a multiple of w_b. ∎

Definition 1 *All $UKP^{c}_{\mathbf{w},\mathbf{p}}$ instances in which $q_1^* \leq 1$ are called **SAW-UKP*** [3].

Remark 1 *We use the name "**SAW**" because of the saw-like shape of the graph of the function $h(w) = w + (p_1 - w_1)\left\lfloor \frac{w}{w_1} \right\rfloor$ defined on $[w_1, w_{max}]$ and for $p_1 > w_1$. All instances of a **SAW-UKP** are given by (w_i, p_i) points from the hypograph $hyp(h)$ ($hyp(h) = \{(w,p) \mid p \leq h(w)\}$).*

The following condition is a necessary condition for $\text{UKP}^{c}_{\mathbf{w},\mathbf{p}}$ to be a **SAW-UKP**.

Lemma 1 *If $UKP^{c}_{w,p}$ is a **SAW-UKP**, then the item 1 is the best one.*

Proof:

$\text{UKP}^{c}_{\mathbf{w},\mathbf{p}}$ is a **SAW-UKP** means that $q_1^* \leq 1$, i.e. for all $i \in I, q_1^i = \frac{p_i - p_1\left\lfloor \frac{w_i}{w_1} \right\rfloor}{w_i - w_1\left\lfloor \frac{w_i}{w_1} \right\rfloor} \leq 1$.

Then we can derive for all $i \in I$:

$\frac{p_i - p_1\left\lfloor \frac{w_i}{w_1} \right\rfloor}{w_i - w_1\left\lfloor \frac{w_i}{w_1} \right\rfloor} \leq 1 \Leftrightarrow (p_i - w_i) \leq (p_1 - w_1)\left\lfloor \frac{w_i}{w_1} \right\rfloor$ which implies

$(p_i - w_i) \leq (p_1 - w_1)\frac{w_i}{w_1} \Leftrightarrow \frac{p_i}{w_i} \leq \frac{p_1}{w_1}$. ∎

It can now be established that U_b^* is tighter than U_3 for this family of UKP.

Theorem 3 *If $UKP^{c}_{\mathbf{w},\mathbf{p}}$ is a **SAW-UKP**, then $U_b^* = U_{\tau^*} \leq U_v \leq U_3$*

Proof: It is assumed that the first three items are of the largest ratio, and also that $\frac{p_3}{w_3} \geq 1$ (as above, if it is not the case, changing $\mathbf{p}$ to $\psi\mathbf{p}, \psi > \max\{\frac{w_1}{p_1}, \frac{w_3}{p_3}\}$ achieves the goal).

According to lemma 1, the item 1 is the best one. It is easy to see that in this case $U_b^* = U_{\tau^*}$. Because of theorem 1 and the relation $U_3 = \max\{U^0, \bar{U^1}\}$, it is enough then to prove that $U_v \leq U^0$. Since 1 is supposed to be the lightest item, we have $w_2 \geq w_1$ and $\left\lfloor \frac{c \bmod w_1}{w_2} \right\rfloor = 0$. Thus $z' = \left\lfloor \frac{c}{w_1} \right\rfloor p_1$ and $c' = \bar{c} = c \bmod w_1$.

$$
\begin{aligned}
U^0 &= \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + \left\lfloor c' \frac{p_3}{w_3} \right\rfloor = \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + \left\lfloor (c \bmod w_1)\frac{p_3}{w_3} \right\rfloor \\
&\geq \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + (c \bmod w_1) = \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + c - \left\lfloor \frac{c}{w_1} \right\rfloor w_1 = \left\lfloor \frac{c}{w_1} \right\rfloor (p_1 - w_1) + c \\
&\geq U_v
\end{aligned}
$$

[3] This definition was first given in [15].

3.1 Summary of upper bounds relations

We summarize here the relations between the bounds just given (U_b^*, U_{τ^*}) and the previously known bounds U_s, U_3 and U_v. These relations are to be taken into account in the computational section 5, where an experimental justification of the solver EDUK2 is presented.

(1) **SAW-UKP** : $U_{\tau^*} = U_b^* \leq U_v$
   (a) **SS-UKP** ($\alpha = 0$) : $U_b^* = U_s = U_3 = U$
   (b) **SC-UKP** and $\alpha > 0$: $\begin{cases} \text{if } \min\limits_{i \in I/\{1\}} \left\lfloor \dfrac{w_i}{w_1} \right\rfloor = 1 : U_b^* = U_s \\ \text{if } \min\limits_{i \in I/\{1\}} \left\lfloor \dfrac{w_i}{w_1} \right\rfloor > 1 : U_b^* < U_s \end{cases}$

(2) **Non-SAW-UKP** (**SC-UKP** with $\alpha < 0$ being in this class) : $U_b^* \overset{>}{<} U_3$ (i.e. these bounds can by in any relation)

Example 1 (A Saw UKP where $U_{\tau^*} < U_v < U_3$) *n=7; c=2900;I={1;...;7};* $\mathbf{p}$*=[300;580;301;601;605;322;310];* $\mathbf{w}$*=[120;245;130;260;310;194;190].*

We can compute that $\mathbf{q}$= [_; -4.; 0.1; 0.05; 0.0714285; 0.297297; 0.142857] (remember that q_1^1 is not defined). Hence $q_1^* \approx 0.297$ and *Example 1* is therefore a **SAW-UKP**. The bounds are: $U_{\tau^*} = U_b^* = 7205 < U_v = 7220 < U_3 = 7246$. The optimal value is 7202.

Example 2 (A non-SAW-UKP with $U_b^* < U_3$) *n=3; c=2900;* $\mathbf{p}$*=[119;297;309];* $\mathbf{w}$*=[119;120;131]. The second item is the best one. We obtain* $q_b^* = 1.090909$ *and* $U_b^* = 7149 < U_3 = 7161$. *The optimal value is* 7140.

Example 3 (A non-SAW-UKP with $U_b^* > U_3$) *n=3; c=63;* $\mathbf{p}$*=[17;30;40];* $\mathbf{w}$*=[15;20;25]. The third item is the best one. We obtain* $\mathbf{q}$= $[\frac{3}{2}; \frac{17}{15}; \_;]$ *and therefore* $q_b^* = \frac{3}{2}$. *We compute that* $U_b^* = 99 > U_3 = 97$. *The optimal value is* 90.

4 Main components of the proposed algorithm

The algorithm described below is based on a convenient combination of two basic approaches used in UKP solvers, namely dynamic programming **(DP)** and branch and bound ***(B&B)*** methods.

Dynamic programming (DP)

One of the recursions [6] used for solving UKP is

$$f(N, y) = \max_{j \in J_y} \{f(N, y - w_j) + p_j\} \text{ for } J_y \subseteq I \text{ and } y \in [w_{\min}, c], \tag{6}$$

where $w_{\min} = \min\{w_i, i \in I\}$.

The eligible set J_y is supposed to contain at least one item i s.t. $x_i > 0$ in some optimal solution to $\text{UKP}^{y}_{\mathbf{w},\mathbf{p}}$. The cardinality of this set is crucially important for the efficiency of any algorithm based on formula (6). To the best of our knowledge EDUK [1] is the only solver that uses this recursion with obvious efficiency. The main components of its implementation are the computation of (6) by slices, a sparse representation of the iteration space, and the use of threshold dominance. Slices are defined as intervals of y, and the sparse representation is based on the particular form of the function f. It is well known that $f(N, y))$ is an increasing stepwise function on y, and can be totally recovered when all skip-points $\{(y, f(N, y))\}$ are known (in the sequel, the couples $\{(y, f(N, y))\}$ will be called *optimal states*.)

The *periodicity* property has been described by Gilmore and Gomory [7] as the capacity y^*, called the *periodicity level*, such that for each $y > y^*$, there is an optimal solution with $x_b > 0$. It is well known that, for each $\text{UKP}^{\infty}_{\mathbf{w},\mathbf{p}}$ such a y^* exists, but its value is not easily detectable. So, although the periodicity property can drastically reduce the search space, it can only be detected in a DP framework. In EDUK this is realized by discovering a capacity $y^+ > y^*$ such that $y^+ = \min\{y | \forall y' \in [y - w_{\max}, y]$ there is an optimal solution of $\text{UKP}^{y'}_{\mathbf{w},\mathbf{p}}$ with $x_b > 0\}$.

Finally, the fact that **DP** algorithms compute optimal solutions for all values of y below the capacity c allows the recursion to be stopped when the capacity $\min\{\max\{\frac{c}{2}, w_{\max}\}, y^+\}$ is reached.

Thanks to all above mentioned properties, in practice, EDUK behaves significantly better than the worst case complexity $O(nc)$ of recurrence (6).

Branch-and-bound (B&B)

Unlike DP, *B&B* algorithms compute an optimal solution *only* for a given capacity, and are dependent on the quality of the computed upper bounds. The MTU2 algorithm proposed by Martello and Toth [10] uses the upper bound U_3 and the now well known *variable reduction scheme*: let z be the objective function value of a known feasible solution, and let $\mathcal{U}$ be an upper bound of $f(N, c - w_j) + p_j$; if $\mathcal{U} \leq z$, then either z is optimal or x_j can be set to zero. We say in this case that item j is "fathomed by bounds".

Hybridization of DP and B&B

There are several complementary ways to integrate a bounds knowledge into a DP.

(1) The first approach is to use the variable reduction scheme in a pre-processing stage to reduce the set N.

(2) The second approach consists in computing, for each *optimal state* $(y, f(N, y))$, an upper bound $U(c - y)$ for a knapsack with $c - y$ capacity. If

$$U(c - y) + f(N, y) \leq z, \tag{7}$$

where z is the incumbent objective value, then the state can be discarded. We say in this case that the state is "fathomed by bounds in a *B&B* context". This *states reduction scheme* (called here *DP with states fathomed by bounds*) significantly reduces the number of states during a sparse representation of the iteration space.

(3) The third approach consists in solving an UKP^{c}_{core} using a *B&B* algorithm in which the *core* set is a subset of the items with the best ratios. If $f(core, c) = U(c)$ then the problem is solved. Otherwise, $f(core, c)$ is used as a value of a known feasible solution during the *DP with states fathomed by bounds* stage.

4.1 The EDUK2 algorithm outline

The algorithm EDUK2 , given below, is an hybridization of EDUK with *B&B* components, according the above given integrations. The basic steps of EDUK2 are:

step 1 Detect in $O(n)$ time the best item b, and find an initial feasible solution with value z. Discard from N all items multiply dominated by b. This is also done in linear time.

step 2 For the reduced set of items N, compute an upper bound U by the techniques described in section 3. Apply the variable reduction scheme in $O(|N|)$ time. Then, select a subset containing the C items with the best ratios (core of size C).

step 3 To improve the lower bound, run a *B&B* algorithm on the core, limiting it to explore no more than B nodes.

step 4 Run *DP with states fathomed by bounds* (see section 4.1.1).

Remark 2 *In the current implementation of EDUK2, we use a B&B similar to the one in MTU1 (Martello and Toth [10]), but it is further enriched with the ability to choose the computed upper bound (currently U_v, U_{τ^*} or U_3). The parameters, B and C, were experimentally tuned and fixed to $C = \min\{n, \max\{100, n/100\}\}$ and $B = 10000$.*

4.1.1 DP with states fathomed by bounds

An enhanced version of EDUK operates in step 4. Its pseudo code is given in listing 1. The function `dp-solve(states,items,ya,yb)` is a dynamic programming based on recurrence (6). It traverses the search space by slices of size `h` [4].

[4] we use `h =` $w_{\texttt{min}}$ but this is a parameter of the algorithm.

Starting from some initial lists of states `states`, and items `items`, `dp-solve` uses threshold dominance to build *dominances free* lists `(states',item')` of items and states with weights in the capacity interval $]ya..yb]$. This part of the program corresponds to the original `EDUK`.

Furthermore, and according to the second integration approach given above, the function `fathoming` applies the variables/states reduction schemes to eliminate all fathomed states and items, returning as result the lists `(states",item")`. These computations may improve the incumbent objective value z. To take this into account, the function `fathoming` proceeds in the following manner: for any unfathomed state $(y, f(N, y))$, a greedy solution of the knapsack UKP$^{c-y}_{states''}$ is found, and completed with the solution of $(y, f(N, y))$. The value of this new feasible solution replaces the old one, if its value, say z', is better than z. This functionality of the DP phase is new and specific for `EDUK2` only.

Note that computing all optimal states $(y, f(N, y))$ with $y \leq \frac{c}{2}$ is enough [5], since any knapsack with capacity $y \in]\frac{c}{2}, c]$ can be solved by completing the solution of UKP$^{y-c/2}_{\mathbf{w},\mathbf{p}}$ with the one of UKP$^{c/2}_{\mathbf{w},\mathbf{p}}$.

5 Performance evaluation experiments

Computational experiments were run in order to: (i) test the efficiency of the *B&B*/DP pairing and the state discriminating capacity of the new bounds U^*_b; (ii) exhibit some actual hard instances. Unfortunately, very few real-life instances of **UKP** have been reported in the literature. For this reason we concentrated our efforts on a set of benchmark tests using: (a) random profit and/or weight generation with some correlation formulae; (b) hard data sets that were specially designed for the *B&B* approach [5].

The main rules for generating interesting (fair) instances are briefly sketched below:

(1) Instances without simple dominance (**wsd**). These are instances with mutually non equal weights and if $w_i < w_j$ then $p_i < p_j$ for all couples (i, j). Thus for instances with integer data $n \leq w_{max} - w_{min} + 1$. This could cause problems with generating large size instances, due to arithmetic overflow and needs special purpose compilers (as the one used for EDUK2).

(2) Instances without collective dominance (**wcd**). One can easily prove that a sufficient condition for an instance to be of type **wcd** is the same as above but with p_i and p_j changed to p_i/w_i and p_j/w_j, respectively (increasing profit/weight ratios on increasing items' weights). A special subclass is the previously mentioned SC-UKP with $\alpha < 0$ (see paragraph 5.1.1.2 and formula

[5] this test was not implemented in `EDUK`

Listing 1. Pseudo code of the dynamic programming with bounds (**step 4**)

```
(*Input:
  items: the remaining set of items;
  states: the list of optimal states with weights ≤y;
  y: the already reached capacity;
  c: the target capacity;
  z: the incumbent objective value;
  u: the upper bound.
*)
(*Output:
    an optimal solution z'
*)
(* Initialization *)
  ya := y;
  yb := y + h;
while (|items'| > 1) and (ya<c/2)  and (z < u) do
   (states',items') := dp-solve(states,items,ya,yb);
   (states'',items'',z') := fathoming(states',items',z);
   ya:=yb;
   yb:=yb+h;
   states:=states'';
   items:=items'';
   if (z<z') then z:=z';
done;
  if (|items'| = 1) then
    stop, return the optimal solution build
    by aggregating the single item  with the
    appropriate element from states'.
  else if (ya >= c/2) then
    stop, return an optimal solution obtained by the
    aggregation of the optimal state of weight yb and
    the one of weight (c-yb).
  else if (z = u) then
     stop, return z.
```

(8)), the SC-UKP subclass, called hard Chung examples (figures 2 and 3) and Table 1, part 3, and also formula (9).

In all runs, the instances solved are of **wsd** type and those reported in figures 2 and 3, and in Table 1 part 3 are of type **wcd**.

Remark 3 *All problems reported below are with integer data although the users of EDUK2 are not restricted to this class only.*

The solver EDUK2 is based on a combination of DP approach and B&B approach to UKP. The main goal of the computational experiments is to check (experimentally) if such hybridization helps. The contestants chosen are EDUK- pure DP based solver which we believe is worthwhile to compete with, and MTU2-B&B based solver with an almost classical good reputation. Competition with CPLEX is added for completeness.

As for the bounds U_3 and U_b^*, we did not notice statistically meaningful inclination in favor of one or the other on a large set of randomly generated instances except for the **SAW-UKP** class. That is why their influence is reported for this class only, while for **non-SAW-UKP** instances we present only the results obtained by using U_b^*.

Very few UKP solvers are available for comparison with `EDUK2`. For example, Babayev *et al.* have proposed an integer equivalent aggregation and consistency approach (CA) that appears to be an improvement over MTU2 [2]. However, this code is not available to us. Caccetta & Kulanoot [4] have recently described two specialized algorithms for solving two particular classes of UKP: `CKU1` for Strongly Correlated UKP (`SC-UKP`) and `CKU2` for Subset Sum Problem(`SS-UKP`). However, these algorithms are not applicable to the general UKP. Thus, we chose to compare `EDUK2` with the only two publicly available solvers: `EDUK` [1], which is considered to be the most efficient DP algorithm [12], and `MTU2`, a well-known *B&B* solver [10].

We start by a comparison of the behaviors of `MTU2`, `EDUK` and `EDUK2` on classic data sets, then we focus on comparing `EDUK` with `EDUK2` on new hard instances not solvable by `MTU2`. In the case of SAW UKP, we study the impact on the resolution time when using the new bound U_{τ^*} instead of U_3. We also compare `EDUK2` with the general purpose solver CPLEX.

`EDUK2` and `EDUK` were written in objective CAML 3.08. The respective codes were all run on a Pentium 4, 3.4GHZ with 4GB of RAM, and the time limit for each run was set to 300 sec. MTU2 was executed on the same machine and compiled with `g77-3.2`. The impact of the bounds was tested by simply substituting the bound U_b^* in `EDUK2` with U_3 in a version called $\texttt{edu}_{U_3}$.

5.1 Classic data sets

A complete study of the classic UKP benchmarks, where the behaviors of `EDUK` and `MTU2` have been compared, can be found in [1]. Most of these UKP appear to be easy solvable by `EDUK2`, and for this reason we report only the most interesting subset of the data from our computational results.

5.1.1 Known "hard" instances

First, we focus on the data sets found to be difficult for MTU2 or EDUK [1].

5.1.1.1 (SS-UKP) The SS-UKP instances ($w = p$) are known to be difficult for EDUK. We built such instances by generating 10 instances for each possible combination of $w_{\min} \in \{100, 500, 1000, 5000, 10000\}$, $w_{\max} \in \{0.5\times10^5; 10^5\}$ and $n \in \{1000; 2000; 5000; 10000\}$ with c randomly generated within $[5\times10^5, 10^6]$. We obtain in this manner 400 distinct instances. The average CPU time for the different algorithms was:

EDUK2: $0.045s$; EDUK: $0.474s$; MTU2: $0.136s$.

According to these results, *EDUK2 is 10 (resp. 3) times faster than EDUK (resp. MTU2)*. The impact of U_b^* with respect to that of U_3 is negligible.

We also tested the sensitivity of the algorithms with respect to $w_{\min}$, and the results showed that EDUK2 is much less sensitive to $w_{\min}$ than EDUK. On an average the time for EDUK increased about 80 times when $w_{\min}$ passed from 100 to 10000, while for EDUK2 the average increase is 40 [6] .

	EDUK2	EDUK	MTU2
$w_{\min} = 100$	0.005s.	0.025s.	0.042s.
$w_{\min} = 10000$	0.2s.	1.82s.	0.25s.

5.1.1.2 (SC-UKP) A set of instances of a *Special SC-UKP* was built according to the formula

$$w_i = w_{\min} + i - 1 \quad \text{and} \quad p_i = w_i + \alpha \qquad \text{with } w_{\min} \text{ and } \alpha \text{ given.} \tag{8}$$

Chung *et al.* [5] have shown that solving this problem is difficult for *B&B* . We set $w_{\min} = 1 + n(n+1)$ and $n \in \{50; 100; 200; 300; 500\}$, and used both a negative and a positive value for α. For each set, we generated 30 instances with a capacity taken randomly from the interval $[10^6, 10^7]$.

$\alpha > 0$ (SAW-UKP) The average time needed to solve the 150 instances was:

EDUK2: 3.32s, edu_{U_3}: 3.37s; EDUK: 4.29s.

MTU2 was able to solve only 9 of the 60 instances with $n \in \{50; 100\}$ and none for $n > 100$.

$\alpha < 0$ (Non-SAW-UKP) The average time for solving the 150 instances was:

EDUK2: 6.01s; edu_{U_3}:5.93s; EDUK:8.65s.

[6] Even more stable behavior is observed for MTU2, but its running time for $w_{\min} = 100$ is 10 times bigger than the one of EDUK2 .

MTU2 was able to solve only 10 of the 60 instances with $n \in \{50; 100\}$ and none for $n > 100$.

From these results, it appears that *EDUK2 is 1.3 (resp. 1.45) times faster than EDUK* when $\alpha > 0$ (resp. <0). We observe that the impact of the new upper bound U_b^* with respect to that of U_3 is negligible. As expected, these instances were hard for MTU2.

Remark 4 *Here we left the U_3 versus U_b^* comparison just as an illustration for their statistical closeness in the case of non-SAW UKP instances.*

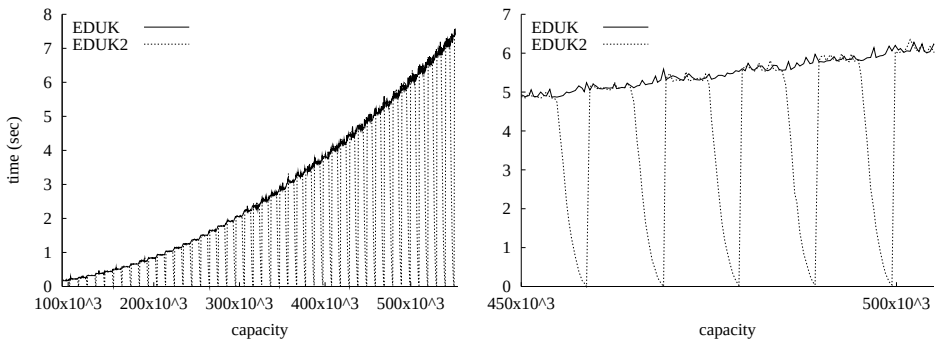
5.1.2 Sensitivity to variations in the capacity: a comparison with EDUK

The *B&B* algorithms are known to be very sensitive to variations in the capacity. DP algorithms, on the other hand, are known to be robust, but their computational time increasing linearly with the capacity value. Our computational experiments show that EDUK2 inherits the good properties of both *B&B* and DP. Data presented in **Fig. 1** were generated by formula (8) as a *Special SC-UKP*. We observe that EDUK2's overall computational time is upper-bounded by the minimum between the time taken by the pseudo-polynomial DP approach and the time for *B&B*. EDUK2 has lost the regular behavior typical of EDUK, but this is in its favor, since the time ratio $\frac{\mathsf{EDUK}(i)}{\mathsf{EDUK2}\,(i)} \geq 1$ is valid for any instance i, and reaches a value of 2.5 for more than 12% of the c values. The local minima in EDUK2's computational time are around points where the capacity is a multiple of the best item's weight. The efficiency of the *B&B* increases near around such capacities (instances) due to the small deviation from 0 of the duality gap (continuous solution is feasible), whose value is known to have a direct impact on the solution time. MTU2 always requires more than 1200 sec., except for 5% of the points where it requires less than 12 seconds. These are the points where EDUK2 finds the solution with the *B&B* (the above mentioned local minima).

*5.1.3 General **SAW-UKP** instances*

This class contains **SAW-UKP** instances generated by the procedure described in Listing 2. Since the generated coefficients p_i satisfy $p_i \leq m_i + p_1 a_i$, $q_1^i = \frac{p_i - p_1 a_i}{m_i}$ and we guarantee that $q_1^* \leq 1$. Moreover $p_i > p_{i+1}$, so there is no simple dominance. 880 instances have been generated in this way using the parameters: $c = \frac{1}{10}\sum w$, $w_{\min} \in \{100; 200; 500; 1000\}$, $w_{\max} \in \{10000; 100000; 1000000\}$ and $n \in \{1000; 2000; 5000; 10000\}$. For each of the 44 possible parameter combinations [7], we randomly generated 20 instances, for which we obtained the following average times:

[7] The combination $n = w_{\max} = 10000$ is not possible due to simple dominance.



Formula (8) where $n = 100$, $w_{\min} = n(n+1)+1$, $\alpha = -3$ and c is randomly and uniformly generated between $[90\,000, 560\,000]$. The whole figure is depicted on the left. On the right, a zoom on the sub-interval $[450\,000, 500\,000]$ is shown. On an average, EDUK2 is more than 25% faster than EDUK.

Fig. 1. Capacity sensitivity of EDUK2 and EDUK

Listing 2. Procedure for generating **SAW-UKP** instances

```
  w_i : randomly generated in strictly increasing order
with the property: w_i mod w_1 > 0, ∀i > 1
  α : a random integer in [1..5]
  p_1 : p_1 = w_1 + α
for i in ]1..n]
  m_i := w_i mod w_1 ;
  a_i = ⌊w_i / w_1⌋ ;
  l_i = 1 + max(p_{i-1}, p_1 × a_i) ;
  p_i : randomly choosen in [ l_i..(m_i + p_1 × a_i)] ;
done;
then pairwise shuffle p and w;
```

EDUK2: 0.129s, edu_{U_3}: 0.252s; EDUK: 0.610s.

We therefore observe that for this family *EDUK2 is about 5 times faster than EDUK*, and using $U_{\tau^*} = U_b^*$ instead of U_3 accelerates EDUK2 by a factor of 2.

Due to arithmetic overflow MTU2 was run with only 200 instances with $w_{\max} = 1000$. For 95 of these instances, it reached the time limit of 300 seconds.

5.1.4 *EDUK2 versus CPLEX versus EDUK*

In this section we compare EDUK2 and EDUK with one of the most popular general purpose mathematical programming optimizers CPLEX of ILOG[8]. For this purpose we focus on three types of problems, each defined by a pair $(\mathbf{w}, \mathbf{p})$ and a wide set of capacities. Each instance has been solved by EDUK2 , EDUK and CPLEX, and the respective required times are reported in **Fig.2-Fig.7**. The first two problems were generated by formula (8) with parameters as given above the graphics. As discussed in section 5.1.1, they are known to be difficult for *B&B*.

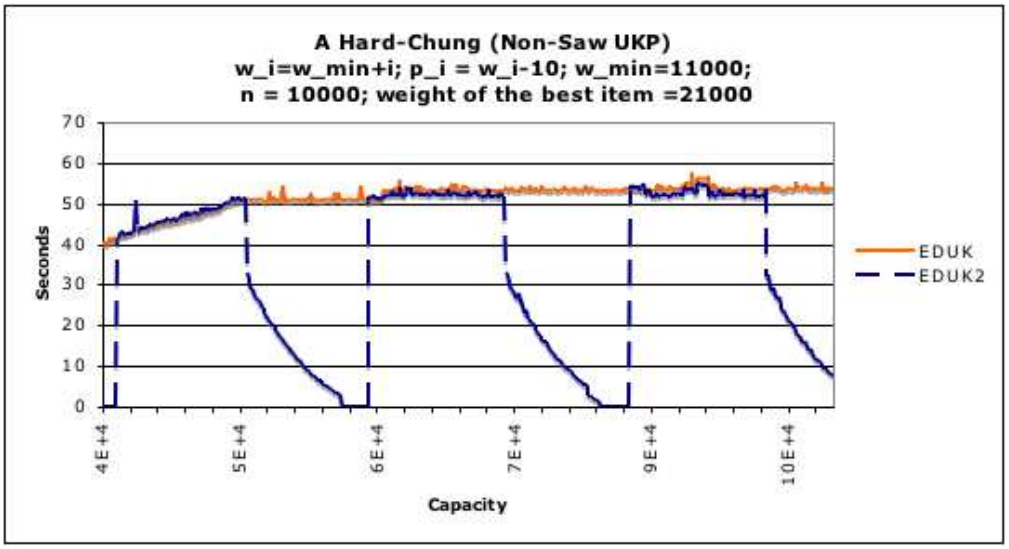


Fig. 2. EDUK2 versus EDUK on a set of 540 hard non-SAW UKP instances

For the first problem, (**Fig.2-Fig.3**), 540 instances were created by uniformly randomly choosing the capacity values in the interval $[4\times10^4,10^5]$. **Fig. 2** compares the behavior of EDUK2 with the one of EDUK. As in **Fig. 1**, EDUK behaves regularly, while the shape of EDUK2's curve permits to distinguish three different cases that alternate periodically: i) a high plateau where both algorithms need the same time since the solution was found by dynamic programming; ii) a low plateau where the solution was found by the bound provided in the B&B phase. EDUK2 computes the results instantaneously being 50 times faster than EDUK. iii) intermediate stage where the solution was found due to B&B/DP hybridization. The weight of the best item (here 21000) is a period of any of these three stages in the behavior of EDUK2.

Next experiment was dedicated to EDUK2 versus CPLEX comparison. Running time for CPLEX was bounded by 600 seconds. **Fig. 3** illustrates that for this lapse of time and on the same data set CPLEX succeeds to solve about 12% of the instances. The solved instances have their capacity in a narrow neighborhood of a multiple of the best item weight. This is clearly seen on **Fig. 3**. These instances correspond in fact to the low plateau ii) above described. In the dominant case, 88%, EDUK2 is

[8] We used version 10.0.1 of CPLEX

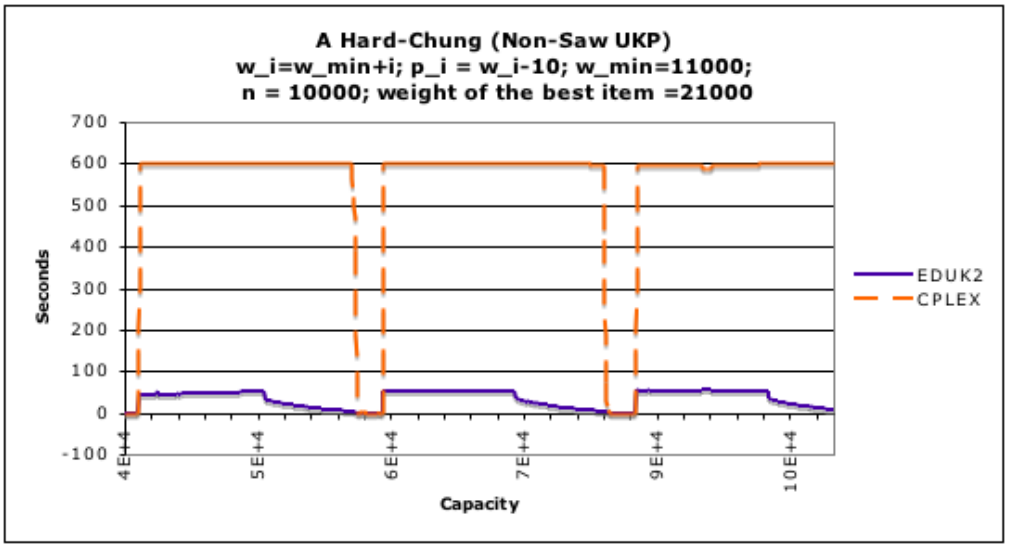


Fig. 3. EDUK2 versus CPLEX on a set of 540 hard non-saw UKP instances

more than 100 times faster than CPLEX.

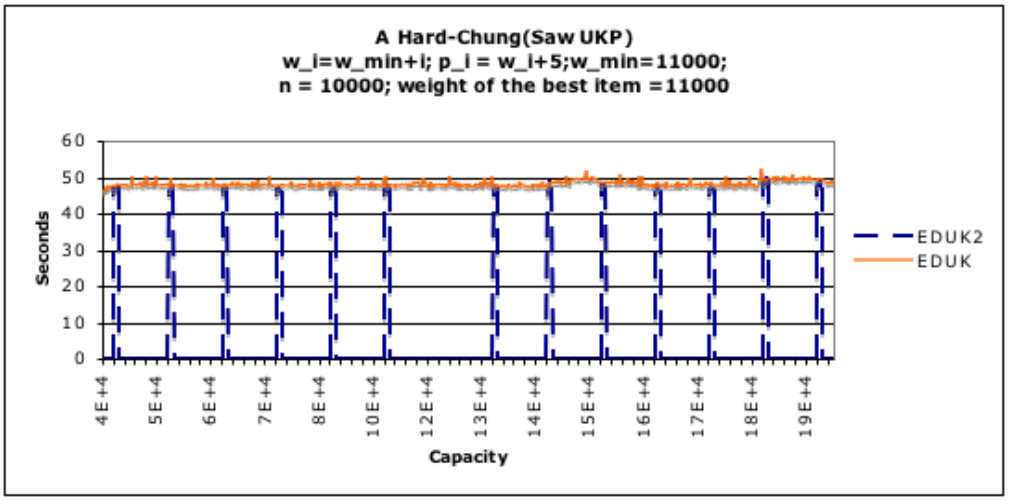


Fig. 4. EDUK2 versus EDUK on a set of 1350 hard saw UKP instances

Figures 4 and 5 illustrate the same comparison in case of SAW UKP instances generated by procedure 2. Here the capacity value is uniformly randomly chosen from the interval $[4 \times 10^4, 2 \times 10^5]$ and 1350 instances were generated in this way. As theoretically expected, due to the new bound, EDUK2 instantaneously finds the solution (except for few values just below a multiple of the weight of the best item). We observe similar phenomena as before: again EDUK2 is about 50 times faster than EDUK (with very few exceptions). CPLEX succeeds to solve about 22% of the instances for the given lapse of time. These instances correspond to a multiple of the best item weight. Outside these rare cases EDUK2 is more than 100 times faster than CPLEX.

Next experiment focusses on randomly generated instances being non-SAW UKP.

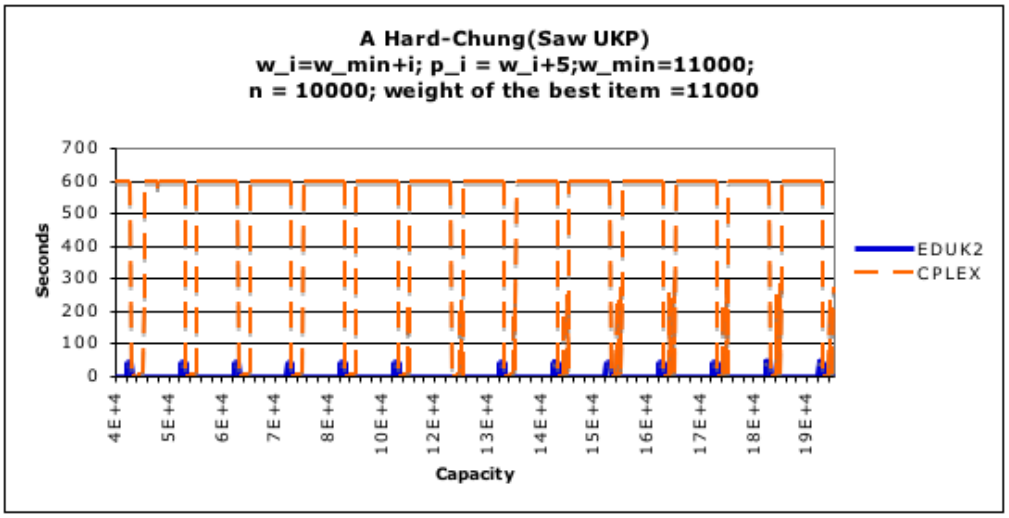


Fig. 5. EDUK2 versus CPLEX on a set of 1350 hard saw UKP instances

We generated 2700 such instances with parameters as described in figures 7 and 6 and a capacity uniformly randomly chosen from the interval $[11 \times 10^4, 43 \times 10^4]$. **Fig. 6** compares EDUK2 versus EDUK on this data set. The behavior of both algorithms is very similar to the one observed on **Fig. 2**: the running time of EDUK2 has a typical saw like shape with minima around the multiples of the best item and upper-bounded by the time of EDUK. **Fig. 7** illustrates EDUK2 versus CPLEX behavior. CPLEX succeeds to solve all instances with a capacity less than 21×10^4 and those with a capacity close to a multiple of the best item, but fails for all other instances with a capacity larger than 21×10^4. For all these instances EDUK2 is as at least 100 times faster than CPLEX.

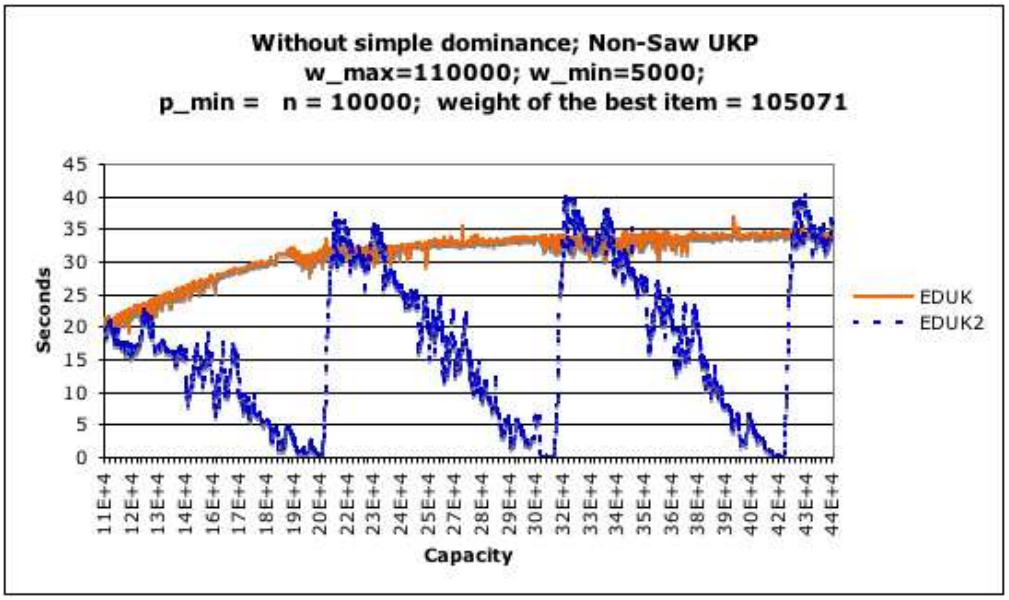


Fig. 6. EDUK2 versus EDUK on a set of 2700 randomly generated UKP instances

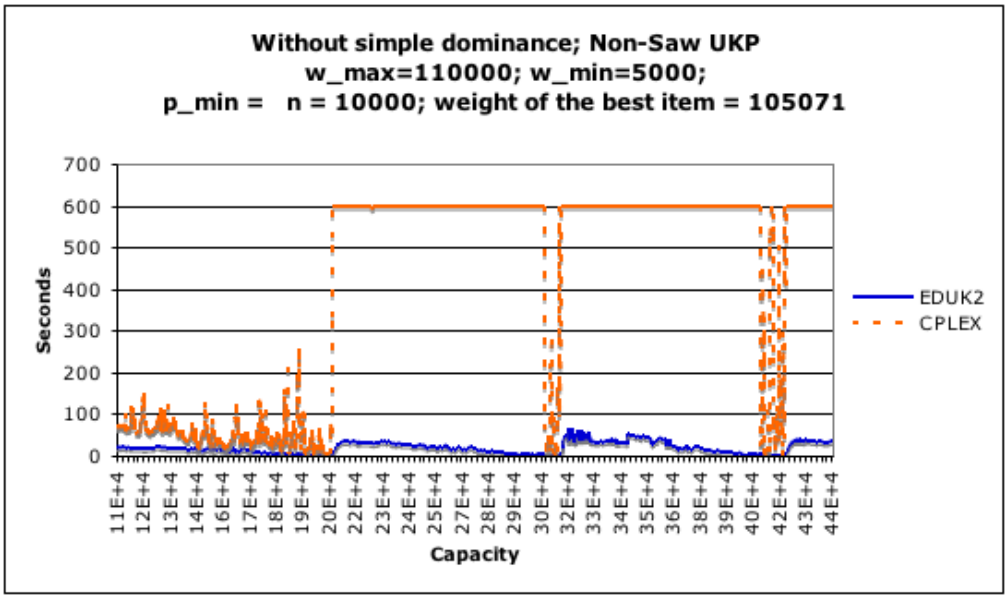


Fig. 7. EDUK2 versus CPLEX on a set of 2700 randomly generated UKP instances

5.2 Do hard UKP instances really exist?

Based on these results, one is inclined to conclude –wrongly– that **UKP** are easy to solve. It is important to remind that, in the above experiments, the considered instances are of moderate size only. A real-life problem of the same size would indeed be easy to solve. However, real problems may have large coefficients, which makes necessary testing the solvers' behavior on such data sets.

5.2.1 New hard UKP instances

In order to construct difficult instances, we considered data sets with large coefficients and/or large number of items. Because `MTU2` cannot be used for such instances because of arithmetic overflow, we restricted our comparisons to `EDUK`, $\texttt{edu}_{U_3}$ and `EDUK2`. For such data sets `EDUK2` and $\texttt{edu}_{U_3}$ benefit of the `num` ocaml library, which provides exact unlimited integer arithmetic to compute the bounds. All the runs were done on a Pentium IV Xeon , 2.8GHZ with 3GB of RAM. CPU time was limited to one hour per instance. If this time limit was reached, we reported 3600 sec. in order to compute the average [9] . We use the notation $x\overline{n}$ to denote $x \times 10^{u+1} + n$, where $0 < \lfloor \frac{n}{10^u} \rfloor < 10$ (e.g. $n = 213, 4\overline{n} = 4213$).

[9] The notation $t(k)$ means that the average time is t sec., with k instances reaching the time limit.

5.2.2 *Instances known to be difficult for B&B*

We generated large data sets using the formula (8). It is easy to see that for such a data set, no more than $w_{\min}$ items are not collectively dominated. For a given n, the formula determines n pairs (w_i, p_i), and we generated 20 different values for c, where c takes random values from $[20\overline{n}; 100\overline{n}]$ (first part in Table 1).

The meaning of the notations used in this table is given in the associated caption. The reported value in the **nmd**, **ncd**, **cpu** columns is the average for the number of instances; the value in the **wdp** columns refers to the total number of instances; the value in the **vrs** [10] , **rp** and **rst** columns, reports the average for the number of instances for which the algorithm enters the DP phase.

EDUK had some trouble in solving these sets and was unable to solve the 20 problems with $\alpha = -5, n = 10^4$, and $w_{\min} = 11 \times 10^4$ in less than one hour. In one special case, where $\alpha = 5$ and $n = w_{\min} = 10000$, the solution was always found immediately in the initial variable reduction step, using the bound U_v. Excluding these two special sets, EDUK2 is on an average from 1.7 to 3.7 times faster than EDUK. Note that for all these instances, the optimal solution was found by EDUK2 and edu_{U_3} either in the variable reduction step, either in the DP phase but never in the *B&B* step. Note that EDUK2 was 1.01 to 1.7 times faster than $\mathsf{edu}_{U_3}$when $\alpha > 0$ (these instances belong to the **SAW-UKP** family). However, in the case of $\alpha < 0$, EDUK2 and edu_{U_3} behave very similarly. For this reason the results of edu_{U_3} are not presented here.

5.2.3 *Data sets with a postponed periodicity level*

For the data in the second part in Table 1, w_i were randomly generated between $[w_{\min}; w_{\max}]$, and p_i values were generated using $p_1 \in [w_1; w_1 + 500]$, $p_i \in [p_{(i-1)} + 1; p_{(i-1)} + 125]$. c was randomly generated between $[w_{\max}; 2 \times 10^6]$. Clearly, for these instances, the number of non-collectively dominated items determines the efficiency of the algorithms. We observed that with this kind of data generation, where $c < 2 \times w_{\max}$ and n is large enough, the periodicity property does not help ($rp \approx 1$). EDUK2 outperforms significantly EDUK and behaves similarly to edu_{U_3}. The results of EDUK2 and EDUK are only given in Table 1.

5.2.4 *Data set without collective dominance*

In order to prevent a DP based solver to benefit from the variable reduction due to the collective dominance, in this section we generate data where the ratio $\frac{p}{w}$ is an increasing function of the weights. We proceeded as follows. w values were

[10] The notation $x(y)$ in this column means that for y instances the optimal value was found in this step and x is the average of the number of reduced variables in the *other* instances.

instance description					EDUK2					edu_{U_3}					EDUK	
20 instances per line					**Hard data sets created using formula (8). c randomly from $[20\overline{n}; 100\overline{n}]$.**											
α	**n**	$\mathbf{w}_{\min}$	**nmd**	**ncd**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**rp**
5	5	10	**n**	**n**	21.77	0(13)	13	0.29	0.047	37.81	642(3)	3	0.38	0.069	80.06	0.108
		15	**n**	**n**	46.57	0(8)	8	0.34	0.099	52.29	83(7)	7	0.56	0.141	111.28	0.188
		50	**n**	**n**	154.19	0(2)	2	0.55	0.470	156.63	0(2)	2	0.68	0.555	261.29	0.661
5	10	10	**n**	**n**	0.03	0(20)	20	-	-	135.22	2420(3)	3	0.54	0.007	336.70	0.008
		50	**n**	**n**	344.12	0(6)	6	0.26	0.037	367.94	0(6)	6	0.41	0.052	915.11	0.079
		110	**n**	**n**	771.53	0(2)	2	0.20	0.112	816.90	0(2)	2	0.26	0.139	2808.50	0.300
-5	5	10	**n**	**n**	64.82	44(6)	6	0.78	0.091						113.67	0.108
		15	**n**	**n**	104.89	11(2)	2	0.61	0.091						183.31	0.188
		50	**n**	**n**	232.26	0(8)	8	0.86	0.650						447.40	0.660
-5	10	10	**n**	**n**	167.26	1317(4)	4	0.67	0.009						317.01	0.009
		50	**n**	**n**	508.37	0(6)	6	0.45	0.058						1539.74	0.079
		110	**n**	**n**	1401.(3)	0(4)	4	-	0.124						(20)	-
200 instances per line					**Data sets with a postponed periodicity level. c randomly from $[w_{\max}; 2 \times 10^6]$**											
n	$\mathbf{w}_{\min}$	$\mathbf{w}_{\max}$	**nmd**	**ncd**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**rp**
20	20	$10\overline{n}$	19985	16851	118.65	11121	2	0.25	0.989						344.81	0.994
50	20	$10\overline{n}$	50000	49999	1026.(1)	28881	0	0.22	1.00						2959.(8)	1.00
20	50	$10\overline{n}$	19999	19924	126.(2)	9955	0	0.23	1.						504.	1
50	50	$10\overline{n}$	50000	49999	1553.(1)	22827	0	0.32	1.00						3289.(51)	1.00
500 instances per line					**Data set without collective dominance (formula (9)). c randomly from $[w_{\max}..1000\overline{n}]$**											
n	$\mathbf{w}_{\min}$		**nmd**	**ncd**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**rp**
5	**n**		**n**	**n**	7.93	3101	23	0.40	0.827						29.05	0.816
10	**n**		**n**	**n**	36.84	5660(1)	13	0.43	0.745						147.76	0.759
20	**n**		**n**	**n**	184.55	12010	3	0.38	0.791						735.24	0.783
50	**n**		**n**	**n**	808.26	25499	2	0.46	1						2764.59	1
					SAW data sets. c randomly from $[w_{\max}; 10\overline{n}]$											
n	$\mathbf{w}_{\min}$	**nbi**	**nmd**	**ncd**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**rp**
10	10	200	9975	1965	8.03	8015	14	0.40	0.597	11.12	5323	2	0.47	0.630	29.06	0.636
50	5	500	49925	5568	70.78	41289(1)	17	0.05	0.51	108.97	25287(1)	11	0.53	0.517	294.30(1)	0.521
50	10	200	49955	8983	71.02	39779(3)	6	0.40	0.49	122.66	26510(3)	3	0.49	0.492	416.88	0.496
100	10	200	99809	6592	264.12	90436	1	0.32	0.510	387.03	65289	1	0.45	0.519	1268.45	0.523

Table 1

Data from **n** and $\mathbf{w}_{\min}$ columns should be multiplied by 10^3 to get the real value. We use the following metrics: **nmd**: number of non-multiply dominated items (step 1 of EDUK2); **ncd**: number of non-collectively dominated items (as computed by EDUK); **cpu**: running CPU time in seconds; **rp**: denotes the ratio $\frac{y^+}{c}$ where y^+ is the capacity level where the algorithm detects that the periodicity level y^* is reached; **vrs**: number of items eliminated in the variable reduction step; **wdp**: number of instances for which the optimal solution was found without using DP (steps 1 to 3); **rst**: ratio of the number of states in the DP phase (step 4 of EDUK2) with respect to the number of states for EDUK.

uniformly and randomly generated within the interval $[w_{\min}..w_{\max}]$ (without duplicates) and were sorted in an increasing order. Then $\mathbf{p}$ was generated using

$$p_1 = p_{\min} + k_1 \text{ and}$$
$$p_i = \lfloor w_i \times (0.01 + \frac{p_{i-1}}{w_{i-1}}) \rfloor + k_i \text{ with } k_i \text{ randomly generated } \leq 10 \quad (9)$$

We set $w_{\min} = p_{\min} = n$, $w_{\max} = 10\overline{n}$, and c was randomly generated within $[w_{\max}..1000\overline{n}]$. We did not observe any significant difference between EDUK2 and edu_{U_3}, though both were about 4 times faster than EDUK (see the associated (third) part in Table 1).

5.2.5 SAW data sets

SAW-UKP instances were generated following procedure 2 with the parameters: $w_{\max} = 1\overline{n}$, $p_{\max} = 2\overline{n}$ and $c \in [w_{\max}; 10\overline{n}]$. For each pair $(n, w_{\min})$, we generated **nbi** *distinct instances* (see the associated (last) part in Table 1).

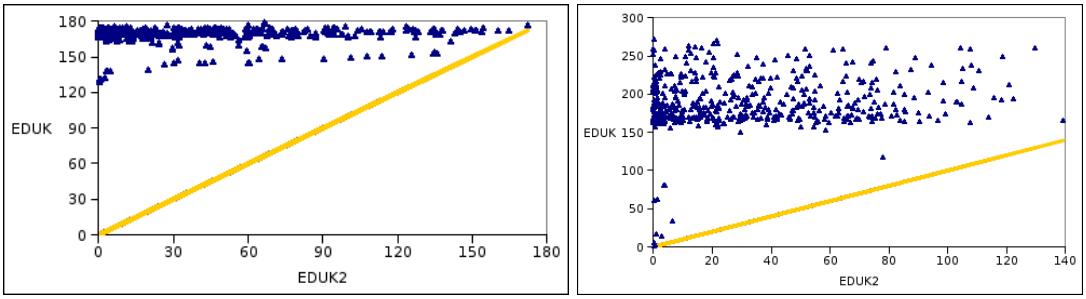
The tight and computationally cheap upper bound for these sets gives a clear advantage to EDUK2 compared to EDUK and edu_{U_3}. The quality of this bound has a noticeable impact on the number of instances solved in the variables reduction step or by the initial *B&B* (column **wdp**), the number of reduced variables (column **vrs**), and the number of states (column **rst**) .

5.2.6 Summary

EDUK2 consistently and significantly outperformed EDUK on all data sets. Once more this is illustrated on **Fig. 8** where the number of points plotted on the left and the right graphics are 2500 each. Any point is an UKP instances of 20000 (left) and of 50000 (right) variables. The average statistics for the running times of EDUK2 and EDUK are: for **SAW-UKP**, generated according listing 2, EDUK2 is 10 times faster than EDUK, while for **non-SAW-UKP** - 3 times. For many instances, EDUK2 yielded the solution immediately while EDUK required several minutes (sometimes more than 1 hour). The efficiency of EDUK2 is obtained by the cumulative effect of the different ways that *B&B* and DP are integrated. Taking into account all the new hard instances (except those generated with formula (8)), the reduction variables step reduces the number of items to be considered on an average varying from 55% to 95%. Integrating bounds during the DP phase further reduces the number of states from 46% to 95%. The impact of the new bound U_b^* is important for all **SAW-UKP** instances and it affects all steps of the algorithm. For the **non-SAW-UKP** instances no significant difference was observed between using U_b^* and U_3.

The superiority of EDUK2 to the general solver CPLEX is (as expected) apparent. In the dominant case, in all tests presented in section 5.1.4 EDUK2 was more than 100 times faster than CPLEX [11] . Additionally to these tests we found useful

[11] CPLEX execution time was upper bounded by 600 sec.



Running times in seconds of EDUK2 (on the horizontal axis) and of EDUK (on the vertical axis). Each point corresponds to one instance. The line is the equal-time line. Left: data set without collective dominance generated by formula (9) with $n = 2 \times 10^4$. Right: **SAW-UKP** data set with $n = 5 \times 10^4$.

Fig. 8. Plots of two large sets of instances

to check the performance of EDUK2 in some recent UKP applications. One such application is described in [16] where CPLEX has been used as UKP solver, instead of a special purpose algorithm. We generated the same set of instances as in [16] for $n = 10^6$. EDUK2 computed 5 such instances on an average time of 0.15 seconds, while the respective running time in [16] is announced to be around 30 hours!

There are still hard instances with large values for n and $w_{\min}$, notably those generated with formula (8), where $\alpha < 0, w_{\min} = 110000, n = 10000$. They were solved by EDUK2 on an average of 25 to 30 minutes. For all these difficult instances, the number of items that are not collectively dominated is very large. Thus, it appears that for such cases, DP algorithm needs to explore a huge iteration space when *B&B* fails to discover the solution.

6 Conclusion

We have shown that a hybrid approach combining several known techniques for solving UKP performs significantly better than any one of these techniques used separately. The effectiveness of the approach is demonstrated on a rich set of instances with very large inputs. The combined algorithm inherits the best timing characteristics of the parents (DP with bounds and *B&B*). We also proposed a new upper bound for the UKP and demonstrated that this bound is the tightest one known for a specific family of UKP. Our EDUK2 algorithm takes advantages of most of the known UKP properties and is able to solve all but the very special hard problems in a very short time. It appears that instances, previously known to be difficult, are now solvable in less than a few minutes.

Acknowledgements Supported by Hubert Curien French-Bulgarian partnership RILA 2006 N^0 15071XF. All computations were done on the Ouest-genopole bioinformatics platform (http://genouest.org). Thanks to N. Malod-Dognin for his help in running CPLEX. The authors would like to thank the two anonymous referees for their insightful comments, corrections and suggestions that significantly improved the paper.

References

[1] R. Andonov, V. Poirriez, and S. Rajopadhye. Unbounded knapsack problem : dynamic programming revisited. *European Journal of Operational Research* , 123(2):168–181, 2000.

[2] D. Babayev, F. Glover, and J. Ryan. A new knapsack solution approach by integer equivalent aggregation and consistency determination. *INFORMS Journal on Computing*, 9(1):43–50, 1997.

[3] V. Boyer, M. Elkihel, and D. El Baz. Efficient heuristics for the 0/1 multidimensional knapsack. In *ROADEF*, pages 95–106. Presses Universitaires de Valenciennes, 2006.

[4] L. Caccetta and A. Kulanoot. Computational Aspects of Hard Knapsack Problems. *Nonlinear Analysis*, 47:5547–5558, 2001.

[5] C-S. Chung, M. S. Hung, and W. O. Rom. A Hard Knapsack Problem. *Naval Research Logistics*, 35:85–98, 1988.

[6] R. Garfinkel and G. Nemhauser. *Integer Programming*. John Wiley and Sons, 1972.

[7] P. C. Gilmore and R. E. Gomory. The Theory and Computation of Knapsack Functions. *Operations Research*, 14:1045–1074, 1966.

[8] S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Manag. Sci.*, 45:414–424, 1999.

[9] S. Martello and P. Toth. A mixture of dynamic programming and branch-and-bound for the subset-sum problem. *Manag. Sci.*, 30(6):765–771, 1984.

[10] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, 1990.

[11] G. L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Willey & Sons, 1988.

[12] U. Pferschy, H. Kellerer, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[13] G. Plateau and M. Elkihel. A hybrid algorithm for the 0-1 knapsack problem. *Methods of Oper. Res.*, 49:277–293, 1985.

[14] V. Poirriez and R. Andonov. Unbounded Knapsack Problem: New Results. In *W*orkshop Algorithms and Experiments (ALEX98), pages 103–111, February 1998.

[15] V. Poirriez, N. Yanev, and R. Andonov. Towards reduction of the class of intractable unbounded knapsack problem. Research report, LAMIH/ROI UMR CNRS-UVHC 8530, 2004.

[16] C. Srisuwannapa and P. Charnsethikul. An exact algorithm for the unbounded knapsack problem with minimizing maximum processing time. *Journal of Computer Science*, 3(3):138–143, 2007.

[17] Nan Zhu and Kevin Broughan. On dominated terms in the general knapsack problem. *Operations Research Letters*, 21:31–37, 1997.