

# Approximated Consistency for Knapsack Constraints <sup>\*</sup>

Meinolf Sellmann

Cornell University  
Department of Computer Science  
4130 Upson Hall  
Ithaca, NY 14853  
sello@cs.cornell.edu

**Abstract.** While global constraints give a broader view on the entire problem and therefore allow more effective constraint propagation, the development of efficient generalized arc-consistency (GAC) algorithms for global constraints is frequently prevented by the fact that the associated decision problems are NP-hard. A prominent example for this is the Knapsack Constraint. On the other hand, there exist approximation algorithms for many NP-hard problems. By introducing the concept of approximated consistency for a special class of global constraints, so-called optimization constraints, we show how existing approximation algorithms can be exploited for the development of efficient filtering algorithms for Knapsack Constraints. As our main result, we show how  $\epsilon$ -GAC for Knapsack and Bounded Knapsack Constraints can be achieved in time  $O(n \log n + \frac{n}{\epsilon^2})$  or  $O(n \log n + \frac{n}{\epsilon^3})$ , respectively.

**Keywords:** global constraints, optimization constraints, cost-based filtering, relaxed consistency, approximation algorithms

## 1 Introduction

When dealing with discrete optimization problems, it is of utmost importance to obtain a global view on the problem that allows to assess what solution quality can still be achieved in a given subtree. To obtain a more global view, in constraint programming it has been suggested to incorporate so-called *optimization constraints* that link the objective function with some other constraints of the problem [2, 4, 6]. For many optimization constraints, efficient filtering algorithms have been developed (see [3, 11, 13, 14] for examples). However, frequently no polynomial time bounded generalized arc-consistency (GAC [1, 7]) algorithm can be developed for optimization constraints, because it is NP-hard to decide whether a *feasible and improving* solution still exists after a variable takes a specific value. We find ourselves in the same unsatisfactory situation when dealing with Knapsack Constraints that are defined as follows:

---

<sup>\*</sup> This work was supported by the Intelligent Information Systems Institute, Cornell University (AFOSR grant F49620-01-1-0076).

**Definition 1.** Let  $n, w_1, \dots, w_n, C, p_1, \dots, p_n, B \in \mathbb{N}$ .  $B$  denotes the objective value to be exceeded,  $C$  the capacity of the knapsack,  $n$  the number of items, and  $w_i$  the weight of item  $i$  with profit  $p_i \forall 1 \leq i \leq n$ . Given  $n$  binary variables  $X_1, \dots, X_n$ , we define:

- The Knapsack Problem consists in maximizing

$$\sum_{i \leq n} p_i X_i \quad \text{s.t.} \quad \sum_{i \leq n} w_i X_i \leq C.$$

- A Knapsack Constraint  $KP(X_1, \dots, X_n, w_1, \dots, w_n, C, p_1, \dots, p_n, B)$  is true, iff

$$\sum_{i \leq n} w_i X_i \leq C \quad \text{and} \quad \sum_{i \leq n} p_i X_i > B.$$

To achieve GAC for a Knapsack Constraint, we have to eliminate all items (i.e. remove value 1 from the corresponding domain) that cannot be part of any feasible solution with profit greater than  $B$ , and we have to permanently include all items (i.e. remove value 0 from the corresponding domain) that are included in all feasible solutions with profit greater than  $B$ . However, since the Knapsack Problem (KP) is NP-hard, so is the problem of achieving GAC for Knapsack Constraints.

Two alternative filtering algorithms for Knapsack Constraints have been proposed in the literature. In [15], Trick develops a pseudo-polynomial time GAC algorithm for Subset-Sum Constraints. These constraints are special Knapsack Constraints where profit and weight of each item are equal. The algorithm uses a dynamic programming scheme for solving the Subset-Sum Problem to optimality and then exploits the information gathered for domain filtering. Addressing the general case where weights and profits can be chosen arbitrarily, Fahle and Sellmann [3] propose to drop the requirement that GAC must be achieved for the Knapsack Constraint. Instead, they introduce a notion of *relaxed consistency* for optimization constraints and use bounds based on linear programming relaxations for polynomial time domain filtering.

While the latter approach is more appealing with respect to the worst-case running time of the filtering procedure, the effectiveness of the algorithm is highly determined by the quality of the bounds that are used. In [3], different filtering algorithms are presented that are based on previously developed integer programming bounds for KP. While these bounds might often be rather tight in practice, their relative error could be arbitrarily close to a factor of 2.

Our aim is to provide filtering algorithms for Knapsack Constraints that are based on bounds with guaranteed accuracy. To achieve this goal, we exploit existing *approximation algorithms* for the Knapsack Problem. With the term “approximation algorithm” we refer to an algorithm that computes a solution to a problem with guaranteed accuracy in polynomial time. A family of approximation algorithms that, for each  $\epsilon > 0$ , provides an algorithm that computes a solution with relative error at most  $\epsilon$  and that runs in time polynomial in the input length and in  $1/\epsilon$ , is called a *fully polynomial time approximation scheme*

(FPTAS). We will show how existing FPTAS for the Knapsack Problem can be used for cost-based filtering with respect to bounds of arbitrary accuracy, whereby the  $\epsilon$ -parameter allows us to trade time for filtering effectiveness.

The remaining presentation is organized as follows: In Section 2, we review the literature on approximation algorithms for KP. Then, in Section 3, we define the notion of approximated consistency for optimization constraints. Finally, in Sections 4 and 5, we develop efficient filtering algorithms for Knapsack Constraints and Bounded Knapsack Constraints.

## 2 Knapsack Approximation

To obtain provably tight bounds on the Knapsack Problem, we can use the existing polynomial time bounded approximation algorithms that solve the KP with arbitrary relative precision  $\epsilon > 0$ . The best currently known FPTAS for KP runs in time  $O(n \log \frac{1}{\epsilon} + \frac{1}{\epsilon^{2+2\delta}})$ , where  $\delta = \frac{\alpha}{1+\alpha}$ , with  $\alpha \in O(C)$  [9]. This result strengthens and is based on the research presented in [5, 8, 12]. We briefly review the main ideas presented in [5] that we will use as a basis for the filtering algorithms that we develop later.

When reviewing the GAC algorithm for Subset-Sum Constraints in [15], we were already reminded that there exist pseudo-polynomial time algorithms for KP that are based on dynamic programming. One of these algorithms is pseudo-polynomial in the optimal objective value  $P^*$ : We set up a matrix  $M$  with  $2P_0+1$  rows and  $n+1$  columns (both starting with index 0), where  $P_0$  is such that  $P_0 \leq P^* \leq 2P_0$ .<sup>1</sup> Now, in  $M_{q,k}$  we store the minimum knapsack capacity that is needed to achieve exactly profit  $q$  when using only items in  $\{1, \dots, k\}$ . Clearly,  $M_{0,k} = 0 \forall k$ , and  $M_{q,0} = \infty \forall q > 0$ , and the following recursion equation holds:

$$M_{q,k} = \min\{M_{q,k-1}, M_{q-p_k,k-1} + w_k\}. \quad (1)$$

By filling the matrix  $M$  row by row from left to right and examining the greatest value  $q$  such that  $M_{q,n} \leq C$ , we can solve the KP in time  $O(nP^*)$ . Now, we can reduce the running time by scaling down the profit values. We set  $\bar{p}_i := \lfloor \frac{p_i}{K} \rfloor$  for some scaling factor  $K \geq 1$  and get a running time in  $O(n\bar{P}^*) = O(n\frac{P^*}{K})$ . It is easy to show that we achieve an  $\epsilon$ -approximate solution for the original problem when we set  $K := \frac{\epsilon P_0}{n}$ , where  $P_0$  is the value of a 2-approximate solution as before<sup>2</sup>. Consequently, we can compute an  $\epsilon$ -approximate solution to KP in time  $O(\frac{n^2 P^*}{\epsilon}) = O(\frac{n^2}{\epsilon})$ .

The running time can be further reduced in  $n$  by partitioning the items into two sets: the set  $L$  of *large* items that contains all items with a profit value greater than some threshold value  $T \geq 1$ , and the set  $S$  of *small* items that contains all items  $i$  with profit  $p_i \leq T$ . We approximate the large item KP by scaling the profit vector and applying the dynamic programming scheme in (1).

<sup>1</sup> The value  $P_0$  can be computed in linear time [8].

<sup>2</sup> Actually, we set  $K := \max\{\frac{\epsilon P_0}{n}, 1\}$  of course, but here and in the following we assume that the scaling factor  $K$  is always greater or equal 1 without further mentioning it.

As a result, for each scaled profit value  $0 \leq q \leq \lfloor \frac{2P_0}{K} \rfloor$ , we get the minimum knapsack capacity  $M_{q,|L|}$  that is needed to achieve profit  $q$  in the scaled large item KP. Now, for all  $q$ , we try to fill the remaining capacity  $C - M_{q,|L|}$  with small items. We do this by inserting the small items in order of decreasing efficiency  $\frac{p_i}{w_i}$  until we reach the first item that exhausts the remaining capacity. We denote the profit that is added by the small items with  $\phi(C - M_{q,|L|})$ . When we take, out of all the  $\lfloor \frac{2P_0}{K} \rfloor + 1$  different knapsacks that were computed, the solution with maximum value  $Kq + \phi(C - M_{q,|L|})$ , it can be shown that the relative error that we make is bounded by  $\frac{K}{T} + \frac{T}{P^*}$ . Therefore, we achieve an  $\epsilon$ -approximation by setting  $K := \frac{\epsilon^2 P_0}{4}$  and  $T := \frac{\epsilon P_0}{2}$  [8]. In order to perform the filling process of the remaining capacities with small items, we can sort the small items according to their efficiency first. Then, the entire algorithm requires time

$$O(n \log n + \frac{P^*}{K} n) = O(n \log n + \frac{4n P^*}{\epsilon^2 P_0}) = O(n \log n + \frac{n}{\epsilon^2}).$$

As stated before, the FPTAS can be strengthened further to give a worst-case running time that is linear in  $n$  for any given constant approximation accuracy  $\epsilon > 0$ . However, the filtering algorithm that we develop later will make use of efficiency and profit orderings. It therefore requires time  $\Omega(n \log n)$  anyway, which is why we make no further effort here to base our filtering algorithm on more sophisticated versions of the general procedure as sketched above.

### 3 Approximated Consistency

Our aim now is to exploit the existing approximation algorithms in order to provide efficient filtering algorithms for Knapsack Constraints. As stated before, we cannot hope to achieve GAC for Knapsack Constraints in polynomial time. Therefore, we introduce a new measure for the consistency of an optimization constraint.

**Definition 2.** Given  $n \in \mathbb{N}$ , let  $X_1, \dots, X_n$  denote some variables with finite domains  $D_1 := D(X_1), \dots, D_n := D(X_n)$ . Furthermore, given a constraint  $\zeta : D_1 \times \dots \times D_n \rightarrow \{0, 1\}$ , and an objective function  $P : D_1 \times \dots \times D_n \rightarrow \mathbb{N}$ , let  $x_i \in D_i \forall 1 \leq i \leq n$ .

- Let  $B \in \mathbb{Q}$  denote a lower bound on the objective  $P$  to be maximized. Then, a function  $\vartheta_{\zeta, P}[B] : D_1 \times \dots \times D_n \rightarrow \{0, 1\}$  with  $\vartheta_{\zeta, P}[B](x_1, \dots, x_n) = 1$  iff  $\zeta(x_1, \dots, x_n) = 1$  and  $P(x_1, \dots, x_n) > B$  is called maximization constraint.
- Given a maximization constraint  $\vartheta_{\zeta, P}[B]$  and some  $\epsilon \geq 0$ , we say that  $\vartheta_{\zeta, P}[B]$  is  $\epsilon$ -GAC, iff for all  $1 \leq i \leq n$  and  $x_i \in D_i$  there exist  $x_j \in D_j$  for all  $j \neq i$  such that  $\vartheta_{\zeta, P}[B - \epsilon P^*](x_1, \dots, x_n) = 1$ , whereby  $P^* = \max\{P(y_1, \dots, y_n) \mid y_i \in D_i, \zeta(y_1, \dots, y_n) = 1\}$ .

Clearly, the Knapsack Constraint is a maximization constraint. Note that our notion of  $\epsilon$ -GAC generalizes the notion of generalized arc-consistency in the sense that GAC is equivalent to 0-GAC. To achieve a state of approximated consistency for a Knapsack Constraint, we must ensure that

1. all items that cannot be part of any feasible solution that achieves a profit greater than  $B - \epsilon P^*$  have to be deleted (i.e. the value 1 must be removed from the corresponding domain), and
2. all items that are included in all feasible solutions with profit greater than  $B - \epsilon P^*$  have to be permanently inserted into the knapsack (i.e. value 0 has to be removed from the corresponding domain).

That is, in contrast to GAC for a maximization constraint, we do not enforce that all domain values are filtered that cannot be used in any improving solution, but at least we want to remove all values for which the performance drops too far below the critical objective value.

## 4 Cost-Based Filtering for Knapsack Constraints

A simple way to achieve a state of approximated consistency for the Knapsack Constraint is to use the algorithm in [9] for probing. This filtering algorithm then runs in time  $O(n^2 \log \frac{1}{\epsilon} + \frac{n}{\epsilon^{2+2\delta}})$  with  $\delta \rightarrow 1$  as  $C \rightarrow \infty$ . In this section, we develop a more sophisticated  $\epsilon$ -GAC algorithm that runs in time  $O(n \log n + \frac{n}{\epsilon^2})$ .

### 4.1 Generalized Arc-Consistency for Knapsack Constraints

The basis of our algorithm is the approximation algorithm described in Section 2. We start by giving a GAC algorithm for the Knapsack Constraint that is based on the dynamic programming scheme in (1). The idea of our algorithm is similar to that described in [15]. We define a weighted, directed, and acyclic graph  $G = (V, E, v)$  by setting

- $V_M := \{M_{q,k} \mid 0 \leq q \leq 2P_0, 0 \leq k \leq n\}$ .
- $V := V_M \cup \{t\}$ .
- $E_0 := \{(M_{q,k-1}, M_{q,k}) \mid k \geq 1, M_{q,k} \in V_M\}$ .
- $E_1 := \{(M_{q-p_k, k-1}, M_{q,k}) \mid k \geq 1, q \geq p_k, M_{q,k} \in V_M\}$ .
- $E_t := \{(M_{q,n}, t) \mid q > B, M_{q,n} \in V_M\}$ .
- $E := E_0 \cup E_1 \cup E_t$ .
- $v(e) := 0$  for all  $e \in E_0 \cup E_t$ .
- $v(M_{q-p_k, k-1}, M_{q,k}) := w_k$  for all  $(M_{q-p_k, k-1}, M_{q,k}) \in E_1$ .

We consider the graph  $G$  because there is a one-to-one correspondence between paths from  $M_{0,0}$  to  $t$  and variable instantiations that yield a profit greater than  $B$ . Moreover, the length of such a path is exactly the weight of the corresponding instantiation. Therefore, every path from  $M_{0,0}$  to  $t$  with length lower or equal  $C$  defines a feasible, improving solution. Vice versa, every feasible, improving solution also defines a path from  $M_{0,0}$  to  $t$  with length lower or equal  $C$ .

The algorithm proceeds as follows: We perform a shortest-path computation on  $G$  and get the shortest-path distances from  $M_{0,0}$  to all other nodes as a byproduct. If the minimum distance from  $M_{0,0}$  to  $t$  is greater than  $C$ , then there exists no feasible, improving solution, and we can backtrack.

Otherwise, following an idea presented in [6], we now reduce  $G$  by eliminating all arcs that cannot be part of any path from  $M_{0,0}$  to  $t$  with length lower or equal  $C$ . We can do this efficiently by computing the shortest-path distances to  $t$ . Then:

- Remove all edges  $(M_{q,k-1}, M_{q,k}) \in E_0$  for which

$$\text{length}(M_{0,0}, M_{q,k-1}) + \text{length}(M_{q,k}, t) > C.$$

- Remove all edges  $(M_{q-p_k, k-1}, M_{q,k}) \in E_1$  for which

$$\text{length}(M_{0,0}, M_{q-p_k, k-1}) + w_k + \text{length}(M_{q,k}, t) > C.$$

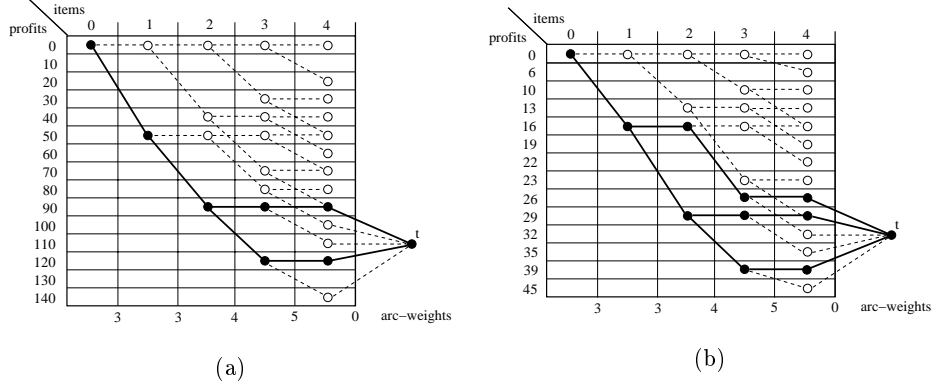
Denote the reduced arc sets with  $E_0^R$  and  $E_1^R$ , respectively. To perform cost-based filtering, we now examine all items  $1 \leq k \leq n$  sequentially. Item  $k$  is removed from the knapsack (i.e., value 1 is filtered from the domain  $D_k$ ), iff there exists no  $q$  such that there is an arc in  $E_1^R$  that ends in  $M_{q,k}$ . Analogously, item  $k$  is added to the knapsack (i.e., value 0 is filtered from the domain  $D_k$ ), iff there exists no  $q$  such that there is an arc in  $E_0^R$  that ends in  $M_{q,k}$ .

The algorithm sketched above is correct and achieves a state of generalized arc-consistency:

- **Correctness:** Assume our algorithm removes value 0 (or value 1) from some domain  $D_k$ . Then, there exists no path from  $M_{0,0}$  to  $t$  in  $G$  with length lower or equal  $C$  such that an arc  $(M_{q,k-1}, M_{q,k}) \in E_0$  (or an arc  $(M_{q-p_k}, M_{q,k}) \in E_1$ , respectively) is visited. Therefore, there also exists no feasible and improving solution such that  $X_k = 0$  ( $X_k = 1$ ).
- **GAC:** Assume that, when setting  $X_k := 0$  (or  $X_k := 1$ ), there exists no extension to a full instantiation of the variables that is feasible with respect to the Knapsack Constraint. Then, there exists no feasible and improving solution with  $X_k = 0$  (or  $X_k = 1$ ). Consequently, there also exists no path from  $M_{0,0}$  to  $t$  in  $G$  with length lower or equal  $C$  that visits an arc  $(M_{q,k-1}, M_{q,k}) \in E_0$  (or an arc  $(M_{q-p_k}, M_{q,k}) \in E_1$ , respectively).

Regarding the time complexity: since shortest-path computations on directed acyclic graphs can be performed in linear time, the algorithm requires time proportional to the size of  $G$ . Now, the out-degree of each node in  $G$  is bounded by 2. Therefore, the algorithm needs time  $O(|V| + |E|) = O(|V|) = O(|V_M|) = O(|M|) = O(nP^*)$ .

An example is given in Figure 1(a). We consider a Knapsack Constraint with four variables  $X_1, \dots, X_4$  with profits  $p^T = (50, 40, 30, 20)$  and weights  $w^T = (3, 3, 4, 5)$ . The knapsack's capacity is  $C = 10$ , and the profit value to be exceeded is supposed to be  $B = 81$ . We see that the value 0 can be removed from the domains of the variables  $X_1$  and  $X_2$ , because in the reduced arc set there are no horizontal arcs left that end in their corresponding columns. Likewise, value 1 can be removed from  $D_4$ . All remaining values cannot be filtered, because the solutions  $X = (1, 1, 1, 0)$  and  $X = (1, 1, 0, 0)$  are both feasible and they improve upon the value of the incumbent solution.



**Fig. 1.** Both figures show the graph  $G$  that is defined for the GAC algorithm. We assume that all arcs are directed from left to right, whereby the arrows are omitted to improve the readability. The matrix structure that is given corresponds to the dynamic programming schema  $M$ , whereby we do not show most cells, nodes and arcs that cannot be reached from  $M_{0,0}$ , again in order to improve the readability. The node-labels are defined by their row and column number, the sink node  $t$  is marked separately. The value of non-horizontal arcs that cross a vertical line is given under that line, horizontal arcs have weight 0. Hollow nodes and dashed arcs mark those nodes and arcs that are removed by the GAC algorithm, because there exists no path from  $M_{0,0}$  to  $t$  with weight lower or equal  $C$  that visits them.

## 4.2 Scaling of Profits

We achieve a polynomial time algorithm for approximated consistency by scaling the profit space and applying the previous algorithm on the scaled problem. We set  $K := \frac{\epsilon P_0}{n}$ ,  $\bar{p}_i := \lfloor \frac{p_i}{K} \rfloor$  for all  $1 \leq i \leq n$ , and  $\bar{B} := \frac{B - \epsilon P_0}{K}$ , and we apply the GAC algorithm in Section 4.1 on the Knapsack Constraint  $KP(X_1, \dots, X_n, w_1, \dots, w_n, C, \bar{p}_1, \dots, \bar{p}_n, \bar{B})$ .

This procedure is correct and achieves a state of  $\epsilon$ -GAC:

- **Correctness:** Assume a value  $b \in \{0, 1\}$  is removed from some domain  $D_k$  by the GAC algorithm on  $KP(X_1, \dots, X_n, w, C, \bar{p}, \bar{B})$ . Then, for the optimal feasible solution  $\bar{x}$  with  $\bar{x}_k = b$  it holds:

$$\bar{p}^T \bar{x} \leq \bar{B} = \frac{B - \epsilon P_0}{K}.$$

With  $x^*$ , we denote the optimal feasible solution to the unscaled problem with side constraint  $X_k = b$ . It follows

$$p^T x^* - \epsilon P_0 = p^T x^* - Kn \leq K \bar{p}^T x^* \leq K \bar{p}^T \bar{x} \leq B - \epsilon P_0,$$

and therefore  $p^T x^* \leq B$ . Thus, it is justified to remove  $b$  from  $D_k$ .

- $\epsilon$ -**GAC:** Let  $k \in \{1, \dots, n\}$ ,  $b \in \{0, 1\}$ , define  $\bar{x}$  and  $x^*$  as before, and assume  $p^T x^* \leq B - \epsilon P^*$ . Then,

$$\bar{p}^T \bar{x} \leq \frac{p^T \bar{x}}{K} \leq \frac{p^T x^*}{K} \leq \frac{B - \epsilon P^*}{K} \leq \frac{B - \epsilon P_0}{K} = \bar{B}.$$

Consequently, value  $b$  is removed from  $D_k$ .

Regarding the time complexity: Clearly, the dominating step is the call to the GAC algorithm that runs in time  $O(n \frac{P^*}{K}) = O(\frac{n^2}{\epsilon} \frac{P^*}{P_0}) = O(\frac{n^2}{\epsilon})$ .

Let us consider the same example as in Section 4.1. Assume we are given  $\epsilon = 0.1$ , and we determine a value  $P_0 = 120$ . Then, the  $\epsilon$ -GAC algorithm sets  $K = 3$ ,  $\bar{p}^T = (16, 13, 10, 6)$ , and  $\bar{B} = 23$ . The GAC algorithm for the modified knapsack constraint then filters value 0 from  $D_1$  and value 1 from  $D_4$  (see Figure 1(b)). Both is correct as we can see from the comparison with the GAC algorithm on the original knapsack constraint. In contrast to the GAC algorithm, the  $\epsilon$ -GAC algorithm is not able to filter value 0 from  $D_2$ . This is okay, though, because there exists a feasible solution  $X = (1, 0, 1, 0)$  that has profit  $50 + 30 = 80 > 69 = 81 - 120/10 = B - \epsilon P_0$ .

### 4.3 Separation of Items

So far we have not achieved any gains over the brute-force probing method that utilizes the best known approximation scheme for Knapsack Problems. For any given constant approximation guarantee  $\epsilon > 0$ , both algorithms require time quadratic in  $n$ . We try to improve on this by separating the items in the style of [5]. We set  $K := \frac{\epsilon^2 P_0}{8}$ ,  $T := \frac{\epsilon P_0}{2}$ , and define

$$S := \{j_1, \dots, j_{|S|}\} := \{1 \leq i \leq n \mid p_i \leq T\},$$

whereby we assume that the items in  $S$  are ordered with respect to decreasing efficiency, i.e.  $\frac{p_{j_l}}{w_{j_l}} \geq \frac{p_{j_{l+1}}}{w_{j_{l+1}}}$  for all  $1 \leq l < |S|$ . Further, let

$$L := \{i_1, \dots, i_{|L|}\} := \{1 \leq i \leq n \mid p_i > T\}$$

and set  $\bar{B} := B - \epsilon P_0$ , and  $\bar{p}_k := \lfloor \frac{p_{i_k}}{K} \rfloor$  for all  $1 \leq k \leq |L|$ . Similar to Section 4.1, we define a weighted, directed, and acyclic graph  $G = (V, E, v)$  by setting:

- $V_M := \{M_{q,k} \mid 0 \leq q \leq \lfloor \frac{2P_0}{K} \rfloor, 0 \leq k \leq |L|\}$ .
- $V := V_M \cup \{t\}$ .
- $E_0 := \{(M_{q,k-1}, M_{q,k}) \mid k \geq 1, M_{q,k} \in V_M\}$ .
- $E_1 := \{(M_{q-\bar{p}_k, k-1}, M_{q,k}) \mid k \geq 1, q \geq \bar{p}_k, M_{q,k} \in V_M\}$ .
- $E_t := \{(M_{q,|L|}, t) \mid M_{q,|L|} \in V_M\}$ .
- $E := E_0 \cup E_1 \cup E_t$ .
- $v(e) := 0$  for all  $e \in E_0$ .
- $v(M_{q-\bar{p}_k, k-1}, M_{q,k}) := w_{i_k}$  for all  $(M_{q-\bar{p}_k, k-1}, M_{q,k}) \in E_1$ .

To complete the definition, we still need to weight the remaining arcs in  $E_t$ . We do this by setting

$$v(M_{q,|L|}, t) := \min \left\{ \sum_{l=1}^s w_{j_l} \mid s \leq |S|, \sum_{l=1}^s p_{j_l} > \bar{B} - Kq \right\}$$

for all  $0 \leq q \leq \lfloor \frac{2P_0}{K} \rfloor$ , whereby we define  $\min \emptyset := \infty$ .

Again, we observe a correspondence between paths from  $M_{0,0}$  to  $t$  in  $G$  and possible knapsack instantiations. While all combinations of large items are possible, the selection of small items is restricted to collections of items with highest efficiency. Note also that the length of a path from  $M_{0,0}$  to  $t$ , if it is lower than infinity, gives the weight of the corresponding solution  $\bar{x}$ . Then, when we denote with  $u_S$  the small item part and with  $u_L$  the large item part of a vector  $u$ , it holds:

$$p^T \bar{x} \geq K \bar{p}_L^T \bar{x}_L + p_S^T \bar{x}_S > \bar{B}.$$

Therefore, any path from  $M_{0,0}$  to  $t$  with weight lower or equal  $C$  defines a feasible solution with profit greater  $\bar{B}$ .

Given  $\epsilon > 0$ , we propose to use Algorithm 1 to achieve  $\epsilon$ -GAC for the Knapsack Constraint  $KP(X_1, \dots, X_n, w, C, p, B)$ .

- 1: Sort the items according to decreasing efficiency and compute a profit ordering of the items.
- 2: Compute  $P_0$  such that for the optimal solution  $P^*$  it holds:  $P_0 \leq P^* \leq 2P_0$ . Then, set  $\bar{B} := B - \epsilon P_0$ ,  $K := \frac{\epsilon^2 P_0}{8}$ , and  $T := \frac{\epsilon P_0}{2}$ .
- 3: Set up the graph  $G = (V, E, v)$  as defined above and compute the shortest-path distances  $length(M_{0,0}, s)$  for all  $s \in V$ .
- 4: If  $length(M_{0,0}, t) > C$ , then set  $D_k := \emptyset$  for all  $1 \leq k \leq n$  and return.
- 5: Compute the shortest-path distances  $length(s, t)$  for all  $s \in V$ .
- 6: Remove all arcs from  $E_0$  and  $E_1$  that cannot be part of any path from  $M_{0,0}$  to  $t$  with length lower or equal  $C$ . Denote the reduced arc sets with  $E_0^R$  and  $E_1^R$ , respectively.
- 7: For all items  $k \in L$  and  $b \in \{0, 1\}$ , remove  $b$  from  $D_k$  iff for all  $M_{q,k} \in V_M$  there exists no arc in  $E_b^R$  that ends in  $M_{q,k}$ .
- 8: For all  $0 \leq q \leq \lfloor \frac{2P_0}{K} \rfloor$ , iterate over all items  $k = j_r \in S$  in order of increasing profit and compute
 
$$v(q, k, 0) := \min\{\sum_{l \leq s, l \neq r} w_{j_l} \mid s \leq |S|, \sum_{l \leq s, l \neq r} p_{j_l} > \bar{B} - Kq\}$$

$$v(q, k, 1) := \min\{w_k + \sum_{l \leq s, l \neq r} w_{j_l} \mid s \leq |S|, p_k + \sum_{l \leq s, l \neq r} p_{j_l} > \bar{B} - Kq\}.$$
- 9: For all  $k \in S$ , remove  $b \in \{0, 1\}$  from  $D_k$  iff for all  $0 \leq q \leq \lfloor \frac{2P_0}{K} \rfloor$ 

$$length(M_{0,0}, M_{q,|L|}) + v(q, k, b) > C.$$
- 10: Return.

**Algorithm 1:**  $\epsilon$ -GAC Knapsack Filtering Algorithm

**Theorem 1.** *Algorithm 1 is correct and achieves  $\epsilon$ -GAC.*

*Proof.* Define

$$f(x) := K \bar{p}_L^T x_L + p_S^T x_S,$$

and for all  $k \in \{1, \dots, n\}$  and  $b \in \{0, 1\}$  set

$$F_{k,b} := \{y \in \{0, 1\}^n \mid w^T y \leq C, y_k = b\}, \text{ and}$$

$$A_{k,b} := \{y \in F_{k,b} \mid \forall l < |S|, j_l \neq k \neq j_{l+1} : y_{j_l} = 0 \Rightarrow y_{j_{l+1}} = 0\}.$$

Without formal proof, it is easy to see that our filtering procedure removes a value  $b \in \{0, 1\}$  from some domain  $D_k$  (no matter whether  $k \in L$  or  $k \in S$ ) iff  $f(x) \leq \bar{B}$  for all  $x \in A_{k,b}$ .

- **Correctness:** Assume a value  $b \in \{0, 1\}$  is removed from some domain  $D_k$ . Denote with  $\bar{x}$  a vector in  $A_{k,b}$  that achieves a maximum profit, i.e.:

$$f(\bar{x}) := \max\{f(x) \mid x \in A_{k,b}\} \leq B - \epsilon P_0.$$

Now, let  $x^* \in F_{k,b}$  denote any feasible knapsack solution with  $x_k^* = b$ . Then, since  $\sum_{i \in L} x_i^* \leq \frac{p_L^T x_L^*}{T}$ , it holds:

$$p^T x^* = p_L^T x_L^* + p_S^T x_S^* < K(\bar{p}_L^T x_L^* + \frac{p_L^T x_L^*}{T}) + p_S^T x_S^*. \quad (2)$$

Further, we know that

$$f(\bar{x}) \geq f((x_L^*, \bar{x}_S)) = K\bar{p}_L^T x_L^* + p_S^T \bar{x}_S. \quad (3)$$

Subtracting inequality (3) from inequality (2) yields:

$$p^T x^* - f(\bar{x}) < \frac{K}{T} p_L^T x_L^* + p_S^T (x_S^* - \bar{x}_S) \leq \frac{K}{T} 2P_0 + T = \epsilon P_0.$$

Consequently,

$$p^T x^* - \epsilon P_0 \leq f(\bar{x}) \leq B - \epsilon P_0,$$

and therefore  $p^T x^* \leq B$ . Thus, value  $b$  is correctly filtered from  $D_k$ .

- **$\epsilon$ -GAC:** Assume there exist  $b \in \{0, 1\}$  and  $k \in \{1, \dots, n\}$  such that for all  $x \in F_{k,b}$ :  $p^T x \leq B - \epsilon P_0$ . Then, for all  $\bar{x} \in A_{k,b}$  it holds that

$$f(\bar{x}) = K\bar{p}_L^T \bar{x}_L + p_S^T \bar{x}_S \leq p^T \bar{x} \leq B - \epsilon P_0 = \bar{B}.$$

Therefore,  $b$  is removed from  $D_k$ . □

Regarding the time complexity: Step 1 takes time  $O(n \log n)$ , and step 2 then can easily be performed in linear time (see [5]). The computations in steps 3–7 can be performed in time  $O(E) = O(n \frac{2P_0}{K}) = O(\frac{n}{\epsilon^2})$  (compare with Section 4.1). Since we are considering the items in  $\hat{S}$  in order of increasing profit, by using the same analysis as in [3], we can show that the computations in step 8 can be performed in time  $O(2n \frac{2P_0}{K}) = O(\frac{n}{\epsilon^2})$ . Step 9 finally takes time  $O(2n \frac{2P_0}{K}) = O(\frac{n}{\epsilon^2})$ . Note that step 1 needs to be carried out only once when the filtering algorithm is called several times with changing domains  $D_k$ . It follows:

**Theorem 2.** *For a Knapsack Constraint  $KP(X_1, \dots, X_n, w, C, p, B)$ , and for all  $\epsilon > 0$ ,  $\epsilon$ -GAC can be achieved in time  $O(n \log n + \frac{n}{\epsilon^2})$ . For  $\Omega(\log n)$  different calls to the filtering routine with changing domains  $D_k \subseteq \{0, 1\}$ , the algorithm runs in amortized time  $O(\frac{n}{\epsilon^2})$  per call.*

One may ask why our filtering algorithm is not based on the best known approximation algorithm in [9], but uses the rather old approximation schema in [5]. While the slightly different separation of items in [9] and the advanced

scaling scheme in [8] could easily be integrated in our algorithm, for us they do not result in an improved running time. The reason for this is twofold: First, when filtering the items in the small item set in step 8, we make extensive use of efficiency and profit orderings, and their computation takes time  $\Theta(n \log n)$  anyway. Second, the advanced scaling scheme in [8] is proposed in order to reduce the number of large items that need to be considered to find an optimal approximation. However, we cannot reduce the number of large items with respect to optimality considerations, because we are looking for *improving* solutions, but not necessarily for *optimal* ones. Therefore, when filtering the large items in steps 3–7, we need to consider all of them, no matter which scaling scheme is used.

Regarding the practicability of our algorithm, for very large  $n$  and really small  $\epsilon$ , there is clearly a problem with respect to the memory requirements. While in the previously developed FPTAS it is sufficient to store only one column of the matrix  $M$  at a time, we require to store the entire graph  $G$ . Therefore, the memory needed is in  $\Theta(\frac{n}{\epsilon^2})$ . The asymptotic constants can be reduced, however, by using an  $\epsilon$ -approximate solution  $P_1$  instead of the 2-approximation  $P_0$  and setting  $K := \frac{\epsilon^2 P_1}{4}$ . Then, the size of  $M$  can be bounded by  $\frac{4n}{(1-\epsilon)\epsilon^2}$  (instead of  $\frac{16n}{\epsilon^2}$ ), and we can show that, for all  $\frac{1}{2} > \epsilon > 0$ , we achieve  $\epsilon$ -GAC in time  $O(n \log n + \frac{n}{\epsilon^2})$ .

## 5 Approximated Consistency for Bounded Knapsack Constraints

To model more realistic problems, we now would like to rid ourselves of the restriction that all variables must have binary domains. We can generalize the results obtained by considering bounded knapsack constraints where each variable is associated with a domain  $D_k = \{0, \dots, u_k\}$ :

**Definition 3.** Let  $n, w_1, \dots, w_n, C, p_1, \dots, p_n, B, u_1, \dots, u_n \in \mathbb{N}$ .  $B$  denotes the value of the incumbent solution,  $C$  the capacity of the knapsack,  $n$  the number of items, and  $w_i$  the weight of item  $i$  with profit  $p_i \forall 1 \leq i \leq n$ . Given  $n$  variables  $X_1, \dots, X_n$  that can take values in  $D_k = \{0, \dots, u_k\}$  for all  $1 \leq k \leq n$ , we define the Bounded Knapsack Constraint as follows:

$BKP(X_1, \dots, X_n, w_1, \dots, w_n, C, p_1, \dots, p_n, B)$  is true, iff

$$\sum_{i \leq n} w_i X_i \leq C \quad \text{and} \quad \sum_{i \leq n} p_i X_i > B.$$

Note that, even though in the definition we require the variables to have domains that start at 0, this is no real restriction, because if some  $X_k$  is required to take values in  $\{l_k, \dots, u_k\}$ , we can simply set  $B' := B - l_k p_k$ ,  $C' := C - l_k w_k$ , and  $D'_k := \{0, \dots, u_k - l_k\}$  and consider  $BKP(X_1, \dots, X'_k, \dots, X_n, w, C', p, B')$ , whereby now the variable  $X'_k$  takes values in  $D'_k$ .

Generally, to approximate the Bounded (or even the Unbounded) Knapsack Problem, we can follow the same procedure as described in Section 2. To cope

with the large items, it has been suggested to introduce multiple copies for each of them. And, in order to compute the profit gained by the small items, it was proposed to sort the items according to their efficiency and then to try to add  $u_k$  copies of the current item until we reach the first item where this is not possible anymore. Then, it is easy to compute the number of copies of this item that can still be introduced without exceeding the knapsack's capacity.

With respect to cost-based filtering, we also try to follow the procedure given in Algorithm 1 very closely. However, the suggestions on how to treat large and small items cannot easily be adapted. First of all, when introducing multiple copies of the large items, the best we can hope for is the information that a variable cannot take values greater than 0 anymore; or likewise, that a variable must take its maximum value. However, we can never get a result that reduces the domains of a variable without setting it to its minimum or maximum value automatically. This effect is of course due to the fact that all copies of an item are symmetric to each other. This means, if for one of the copies it is found that it has to (or must not, respectively) be included in the knapsack, this automatically holds for all other copies, too.

The other problem that we are facing regards the small items. A simple adoption of the procedure given in Section 4 also gives us some trouble to determine how many copies of an item we can afford to remove from (or to insert in, respectively) our knapsack without losing too much profit. In what follows, we address both problems and show how to tackle them efficiently.

### 5.1 Filtering of Large Items

Let us start by considering the set of large items. Instead of adding  $u_k$  copies for each item, we suggest to add a polynomial number of arcs to the graph defined in Section 4.2. First, we observe that, for any large item  $k \in L$ ,  $u_k$  can be bounded from above, because:  $u_k \frac{\epsilon P_0}{2} = u_k T \leq u_k p_k \leq 2P_0$ , and therefore,  $u_k \leq \frac{4}{\epsilon}$ .

Then, we recall that the edge set was partitioned into the sets  $E_0$ ,  $E_1$ , and  $E_t$ . The last set contains all arcs that end in the sink node  $t$ , whereas the first two sets were used to model the choice between insertion and not-insertion of an item. In the same manner, we can introduce additional arc sets  $E_2, \dots, E_u$  that model the insertion of multiple copies of an item, whereby  $u := \max\{u_{i_k} \mid 1 \leq k \leq |L|\} \leq \frac{4}{\epsilon}$ . Formally, we define:

$$E_l := \{(M_{q-l\bar{p}_k, k-1}, M_{q,k}) \mid k \geq 1, u_{i_k} \geq l, q \geq l\bar{p}_k, M_{q,k} \in V_M\} \quad \forall 2 \leq l \leq u.$$

The newly added arcs are weighted by setting

$$v(M_{q-l\bar{p}_k, k-1}, M_{q,k}) := lw_{i_k} \quad \forall (M_{q-l\bar{p}_k, k-1}, M_{q,k}) \in E_l, 2 \leq l \leq u.$$

With this setting, we are able to consider all possible instantiations to large item variables by conducting shortest-path computations in  $G$ . To perform cost-based filtering, we reduce the graph again in the usual way and check whether there exist  $1 \leq k \leq |L|$  and  $0 \leq l \leq u$  such that  $E_l^R$  does not contain arcs anymore that end in some node  $M_{q,k} \in V$ .

With respect to the worst case running time, we lose a factor of  $\frac{1}{\epsilon}$  because now  $|E| \in O(\frac{1}{\epsilon}|V|)$ . Therefore, the filtering of the large items now takes time  $O(\frac{n}{\epsilon^3})$ .

## 5.2 Filtering of Small Items

Now let us consider the items in the set  $S$ . Recall from Algorithm 1 (steps 8 and 9) that, in order to filter values for small items, for all  $0 \leq q \leq \frac{2P_0}{K}$  we have to find out whether we can still close the profit-gap between  $Kq$  and  $\bar{B}$  with the help of the remaining available capacity  $C - \text{length}(M_{0,0}, M_{q,|L|})$  when a certain variable takes a specific value. If we use the same approach as presented in Algorithm 1, for bounded knapsack constraints this requires time  $\Theta(n \max\{u_k \mid k \in S\})$ . Now, in contrast to the large items, the small item's domains cannot tightly be bounded from above. Therefore, this procedure has pseudo-polynomial/exponential running time.

We can do much better though, and we can even rid ourselves from the necessity to compute a profit ordering of the items: Assume all items in  $S$  (for simplicity, let us assume  $S = \{1, \dots, n\}$ ) are ordered with respect to decreasing efficiency  $e_i := \frac{p_i}{w_i}$  for all  $1 \leq i \leq n$ . We consider the items sequentially. Denote the current item with  $k$ . If there is still capacity in the current knapsack  $X$  (recall from Section 4.2 that we need to consider a sequence of knapsacks) left, we insert  $u_k$  copies of item  $k$ . Let  $s^X$  denote the first item where this is not possible anymore. Then, we add as many copies of  $s^X$  as is still possible; the number of copies of  $s^X$  that are inserted is denoted with  $c_s^X$ . Furthermore, we denote the value that the small items achieve in this way by  $\phi^X(C^X)$ , whereby  $C^X$  denotes the current knapsack's capacity. Likewise, we denote with  $\phi_s^X(W)$  the capacity that the *remaining* items can achieve (whereby at most  $u_{s^X} - c_s^X$  copies of item  $s^X$  are allowed) by exploiting some given capacity  $W$  in the same manner as described for  $\phi^X$ . Now, denote with  $R^X := (u_{s^X} - c_s^X)w_{s^X} + \sum_{i>s^X}^n u_i w_i$  the total weight of the remaining items. Then, for a given profit value  $B^X$  that has to be exceeded by the small items, and for all  $1 \leq k < s^X$ , we define

$$\Delta_k^X := \max\{W \leq R^X \mid \phi^X(C^X) + \phi_s^X(W) \geq B^X + 1 + W e_k\}.$$

With this setting,  $\Delta_k^X$  reflects the total weight of an item  $k$  that we can afford to lose while still achieving a total profit of at least  $B^X + 1$ . Note that this total weight is allowed to exceed the actual weight of all copies of an item  $k$ , which is exactly  $u_k w_k$ . Now, assume  $\Delta_k^X \geq u_k w_k$ . Then, for all item  $k < s^X$  in the current knapsack  $X$ , we can afford to use no copy of item  $k$  at all, and therefore, no reduction of the domain  $D_k$  can take place. However, if  $\Delta_k^X < u_k w_k$  for some  $k < s^X$ , then we cannot afford to lose more than  $\frac{\Delta_k^X}{w_k}$  copies of item  $k$ . Then, we set  $D_k^X := \left\{ \left\lceil \frac{u_k w_k - \Delta_k^X}{w_k} \right\rceil, \dots, u_k \right\}$ .

Likewise, for all  $k > s^X$ , we define

$$\Gamma_k^X := \max\{W \leq C^X \mid \phi^X(C^X - W) \geq B^X + 1 - W e_k\},$$

and, if  $\Gamma_k^X < u_k w_k$ , we set  $D_k^X := \left\{ 0, \dots, \left\lfloor \frac{\Gamma_k^X}{w_k} \right\rfloor \right\}$ .

The important observation is, that there is some monotonicity among the  $\Delta_k^X$  and  $\Gamma_k^X$ . Since the items are ordered with respect to decreasing efficiency, it holds

$$\Delta_{k+1}^X \geq \Delta_k^X \quad \forall 1 \leq k < s^X \quad \text{and} \quad \Gamma_{k+1}^X \geq \Gamma_k^X \quad \forall s^X < k \leq n.$$

Therefore, by using a similar routine to that described in [3], once an efficiency ordering of the items is known, the computation of the different  $D_k^X$  can be done in time  $O(n)$ . To complete the computation, eventually we determine the minimal  $D_{s^X}^X$  separately, which can also be done easily in time  $O(n)$  once the efficiency ordering of the items is known.

After having computed  $D_k^X$  for all  $1 \leq k \leq n$  and for all small item knapsacks  $X$  that need to be considered, we can finally set

$$D_k := \bigcup_X D_k^X.$$

Since there are  $O(\frac{1}{\epsilon^2})$  knapsacks that need to be considered, the entire filtering process for the small items takes time  $O(n \log n + \frac{n}{\epsilon^3})$ .

Putting the results for the large and the small items together, we have shown

**Theorem 3.** *For a Bounded Knapsack Constraint  $BKP(X_1, \dots, X_n, w, C, p, B)$ , and for all  $\epsilon > 0$ ,  $\epsilon$ -GAC can be achieved in time  $O(n \log n + \frac{n}{\epsilon^3})$ . For  $\Omega(\log n)$  different calls to the filtering routine with changing domains of the form  $D_k = \{l_k, \dots, u_k\}$ , the algorithm runs in amortized time  $O(\frac{n}{\epsilon^3})$  per call.*

## 6 Conclusion and Future Work

Since achieving a state of generalized arc-consistency for many global constraints is an NP-hard task, we introduced the notion of approximated consistency for optimization constraints. This notion allows to determine the filtering power of a propagation algorithm by the guaranteed approximation quality of the bounds that are used. Most importantly, by trading time for effectiveness, the  $\epsilon$ -parameter allows to tune the filtering algorithm with respect to the specific constraint optimization problem that has to be solved.

For Knapsack Constraints, we have shown how existing approximation algorithms for the Knapsack Problem can be exploited for the development of efficient filtering algorithms. We presented an algorithm that achieves  $\epsilon$ -GAC for Knapsack Constraints. For all constant  $\epsilon > 0$ , that algorithm runs in linear time for  $\Omega(\log n)$  different calls with changing variable domains. It therefore improves clearly upon the filtering algorithms developed in [3]. Moreover, we developed an extension of our algorithm that can cope with Bounded Knapsack Constraints and that achieves  $\epsilon$ -GAC in amortized time  $O(\frac{n}{\epsilon^3})$ .

The filtering algorithms described in this paper are currently being implemented. We shall soon be able to evaluate their practical performance and to

perform experiments that give an insight regarding good choices of the approximation accuracy. Since we can smoothly vary the filtering effectiveness, we hope that these experiments will eventually establish a better understanding of the frequently observed duality between inference and search.

## References

1. K. R. Apt. The Rough Guide to Constraint Propagation. *5th International Conference on Principles and Practice of Constraint Programming (CP)*, LNCS 1713:1–23, 1999.
2. T. Fahle, U. Junker, S.E. Karisch, N. Kohl, M. Sellmann, B. Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8(1):59–81, 2002.
3. T. Fahle, M. Sellmann. Cost-Based Filtering for the Constrained Knapsack Problem. *Annals of Operations Research*, 115:73–93, 2002.
4. F. Focacci, A. Lodi, M. Milano. Cost-Based Domain Filtering. *Principles and Practice of Constraint Programming (CP)* Springer LNCS 1713:189–203, 1999.
5. O.H. Ibarra, C.E. Kim. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. *Journal of the ACM*, 22(4):463–468, 1975.
6. U. Junker, S.E. Karisch, N. Kohl, B. Vaaben, T. Fahle, M. Sellmann. A Framework for Constraint programming based column generation. *Principles and Practice of Constraint Programming (CP)*, Springer LNCS 1713:261–274, 1999.
7. V. Kumar. Algorithms for Constraints Satisfaction problems: A Survey. *The AI Magazine, by the AAAI*, 13:32–44, 1992.
8. E.L. Lawler. Fast Approximation Algorithm for Knapsack Problems. *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pp. 206–213, 1977.
9. Y. Liu. On the Fully Polynomial Approximation Algorithm for the 0-1 Knapsack Problem. *Theory of Computing Systems*, 35:559–564, 2002.
10. J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. *12th National Conference on Artificial Intelligence, AAAI*, pp. 362–367, 1994.
11. J.-C. Régin. Cost-Based Arc Consistency for Global Cardinality Constraints. *Constraints*, 7(3-4):387–405, 2002.
12. S. Sahni. Approximate algorithms for the 0/1 Knapsack Problem. *Journal of the ACM*, 22(1):115–124, 1975.
13. M. Sellmann. An Arc-Consistency Algorithm for the Weighted All Different Constraint. *8th International Conference on Principles and Practice of Constraint Programming (CP)*, LNCS 2470:744–749, 2002.
14. M. Sellmann, T.Fahle. Coupling Variable Fixing Algorithms for the Automatic Recording Problem. *Annual European Symposium on Algorithms (ESA)*, Springer LNCS 2161: 134–145, 2001.
15. M. Trick. A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. *3rd International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR)*, pp. 113–124, 2001.