

Cost based Filtering for the Constrained Knapsack Problem *

Torsten Fahle, Meinolf Sellmann

Department of Mathematics and Computer Science
Fürstenallee 11, D-33102 Paderborn
{tef,sello}@uni-paderborn.de

February 7, 2001

Abstract

We present cost based filtering methods for Knapsack Problems (KPs). Cost based filtering aims at fixing variables with respect to the objective function. It is an important technique when solving complex problems such as Quadratic Knapsack Problems, or KPs with additional constraints (Constrained Knapsack Problems (CKPs)). They evolve e.g. when Constraint Based Column Generation is applied to appropriate optimization problems. We develop new reduction algorithms for KP. They are used as propagation routines for the CKP with $\Theta(n \log n)$ preprocessing time and $\Theta(n)$ time per call. This sums up to an amortized time $\Theta(n)$ for $\Omega(\log n)$ incremental calls where the subsequent problems may differ with respect to arbitrary sets of necessarily included and excluded items.

Keywords: constraint programming, constrained knapsack problems, optimization constraints, cost based filtering, reduction algorithms.

1 Introduction

An effective way of combining the advantages of Constraint Programming (CP) and Operations Research (OR) techniques is the development of optimization constraints that perform cost based filtering [8]. Optimization constraints are used to include/exclude items that must/cannot be part of any improving solution. We introduce propagation algorithms to perform cost based filtering for Constrained Knapsack Problems (CKPs).

Recently, a new framework for the integration of CP and OR within column generation approaches was developed, the so called Constraint Based Column Generation [13]. It describes a generic way of how to treat arbitrary constraints for the constrained subproblem in the column generation phase. The approach has been successfully used for the Crew Assignment Problem [6], where the subproblem is a Constrained Shortest Path Problem (CSP). Until now, the new paradigm has only been used in combination with CSPs. We show exemplary how CKPs can be used when generating columns for a Constrained Cutting Stock Problem.

*This work was partly supported by the UP-TV project, partially funded by the IST program of the Commission of the European Union as project number 1999-20 751, by the German Ministry for Education and Research Bmb+f (Parpap project 01 HR 9955), and by.

In every tree search, there is a trade-off between the quality of the bounds (i.e. the time saved due to an effective pruning) and the time needed for their computation. When solving pure KPs, a big effort to tighten the problem in every search node usually does not pay off. However, in a CP search the total cost per choice point is usually much bigger. Thus, the gain due to an effective bounding and tightening is higher, and better bounds can be used profitably for pruning and propagation. On the other hand, fast KP reduction algorithms using weak bounds, such as the algorithm developed by Dembo and Hammer [5], are not effective enough in a CP context.

Based on reduction techniques for KP, we develop propagation routines for knapsack constraints. We present several new reduction algorithms using bounds of different quality. The method that we consider the most interesting one theoretically and practically is based on a bound proposed by Martello and Toth in [15]. By reusing information gained in an initial preprocessing step taking time $\Theta(n \log n)$, the actual reduction per choice point only requires linear time. We numerically compare two of the new methods with two other reduction algorithms that have been proposed earlier in the KP literature.

1.1 Constrained Knapsack Problems

The CKP is basically a knapsack problem with additional constraints. We do not require these additional constraints to be linear. Nevertheless, objective function and the knapsack constraint itself have to be linear. Formally, the CKP is defined as follows:

Definition 1.1 *Let $C, n, w_1, \dots, w_n \in \mathbb{N}$; $p_1, \dots, p_n \in \mathbb{Z}$. C is the capacity of the knapsack, n the number of items, and w_i the weight of item i with profit $p_i \forall 1 \leq i \leq n$. Moreover, let $w := (w_1, \dots, w_n)^T$, and $p := (p_1, \dots, p_n)^T$.*

1. Let $\mathbb{B} := \{0, 1\}$, and $G := \{x \in \mathbb{B}^n \mid w^T x \leq C\}$.
2. Let $k \in \mathbb{N}$, and $R := \{r_1, \dots, r_k \mid r_j : \mathbb{B}^n \rightarrow \mathbb{B} \quad \forall 1 \leq j \leq k\}$. Every $r \in R$ is called a (knapsack) rule and R is called a (knapsack) rule set.
3. Every $x \in G$ is called feasible (with respect to a given rule set R), iff $r(x) = 1 \forall r \in R$. $F(R) := \{x \in G \mid x \text{ is feasible}\}$ is called the set of feasible constrained knapsacks (with respect to rule set R). To simplify the notation, we often write F instead of $F(R)$ if R is known from the context.
4. The Constrained Knapsack Problem then is to

$$\text{maximize } p^T x, \quad x \in F.$$

Notice, that for the unconstrained KP it holds $F = G$. Here, we investigate the general case of $F \subseteq G$. Algorithms for the unconstrained KP are not able to solve the CKP, because they do not allow to incorporate additional constraints. Moreover, algorithms designed to solve pure KPs make certain assumptions that do not hold for CKPs. E.g., it is not clear for the CKP that we can require the profits to be non-negative (as it is the case for KP), because the strategy to omit items with positive weight and negative profit [17] may not yield feasible knapsacks at all.

In the following, with identifiers $\mathbb{B}, C, n, w, p, G, R$, and F we refer to the above definition. We will sometimes need to refer to reduced CKPs where an item $i \in \{1, \dots, n\}$ is either

included or excluded in any feasible solution. We refer to those problems with $\text{CKP}[x_i = 1]$ or $\text{CKP}[x_i = 0]$, respectively.

1.2 Applications for Constrained Knapsack Problems

CKPs appear in various application areas. In the following, we sketch three examples:

1.2.1 A Multimedia Application

A CKP accompanied by a special shortest path constraint occurs in the *Automatic Recording Problem (ARP)*. That problem consists of finding an optimal selection of TV broadcasts for video recording. The shortest path constraint ensures that only non-overlapping broadcast can be recorded, whereas the knapsack constraint models the storage limit of the recording device. The objective is to maximize user’s satisfaction. In [22], the ARP has been solved by Lagrangian relaxation using the algorithm presented in Section 3.1. The approach outperforms both, pure CP as well as pure OR methods for the ARP.

1.2.2 Constraint Based Column Generation

When applying the Constraint Based Column Generation paradigm [13] to appropriate optimization problems, CKPs occur as subproblems. As an example, when applying Column Generation to the *Constrained Cutting Stock Problem* – a Cutting Stock Problem with additional constraints on the cutting patterns – results in a CKP subproblem. Additional constraints usually stem from real-world applications (an example for real-world constraints is given in [3]) and may be non-linear.

In the CP-based Column Generation context, we search for feasible knapsacks with negative reduced costs. In the Constrained Cutting Stock Problem, for instance, each cutting pattern has cost 1 since we try to minimize the number of rolls needed to cover the specified demand. Thus, the objective in the subproblem is to minimize $1 - \pi^T x$ (i.e. to minimize the reduced costs of the cutting pattern), where π is the vector of dual values corresponding to the current optimal solution of the continuous relaxation of the master problem. Our objective in the CKP then is to maximize $\pi^T x$ with an initial lower bound of 1.

1.2.3 Fast Reduction Techniques for Quadratic Knapsack Problems

The *Quadratic Knapsack Problem (QKP)* calls for maximizing a quadratic boolean objective function subject to a linear knapsack constraint. The relax and cut algorithm of Porto et al. [21] computes bounds of the QKP by linearizing the problem to KP, then tightening the problem by adding three families of valid inequalities, and finally solving the resulting linear program (LP) by Lagrangian relaxation. Thus, a series of KPs has to be solved in every search node. The authors mention, that fixing variables was vital for their approach. Typically, filtering algorithms for KP are used to reduce the size of the given QKP [2]. The algorithms proposed in Section 3 may help to increase the performance of the overall approach in [21].

1.3 Constrained Knapsack vs. Pure Knapsack Problems

For pure knapsack problems without additional constraints, the state of the art solving techniques would focus on a so called *core problem*, which may be extended during the optimization

process [14, 20]. For these algorithms it is not straight forward to see how the reduction algorithm we present in the following could be integrated efficiently. However, in the context of this paper we focus on Constrained Knapsack Problems, where a branch and bound tree search framework is needed to find feasible solutions with respect to additional constraints, and where algorithms tailored for the pure KP must fail. This motivates the decision to make use of orderings in an algorithm for CKP, although it is known already that the calculation of those orderings often does not pay off when solving pure KPs.

The remaining paper is organized as follows: In Section 2, we formalize the concept of optimization constraints and present upper bounds and reduction techniques for KP from the literature. Section 3 explains the new algorithms for a quick propagation of knapsack constraints. An experimental evaluation of these algorithms, as well as a comparison with alternative approaches is presented in Section 4. Finally, we conclude in Section 5.

2 Preliminaries

2.1 Optimization Constraints

CKPs belong to the class of optimization constraints, i.e. constraints reflecting feasibility and cost aspects simultaneously. Optimization constraints have been addressed e.g. in [8, 9, 13, 6, 19].

Before proceeding with the special case of CKPs, we would like to propose a formal concept of optimization constraints. To our knowledge, in the literature this has not been done before.

Given $n \in \mathbb{N}$, let V_1, \dots, V_n denote variables with finite domains $D(V_1), \dots, D(V_n)$. Further, given a constraint $\zeta : D(V_1) \times \dots \times D(V_n) \rightarrow \{0, 1\}$, and an objective function $Z : D(V_1) \times \dots \times D(V_n) \rightarrow \mathbb{R}$, let $v_i \in D(V_i) \forall 1 \leq i \leq n$.

Definition 2.1 *Let $B \in \mathbb{R}$ denote an upper/lower bound on the objective Z to be minimized/maximized.*

- $\vartheta_{\zeta, Z}[B] : D(V_1) \times \dots \times D(V_n) \rightarrow \{0, 1\}$ with $\vartheta_{\zeta, Z}[B](v_1, \dots, v_n) = 1$ iff $\zeta(v_1, \dots, v_n) = 1$ and $Z(v_1, \dots, v_n) < B$ is called *minimization constraint*.
- $\vartheta_{\zeta, Z}[B] : D(V_1) \times \dots \times D(V_n) \rightarrow \{0, 1\}$ with $\vartheta_{\zeta, Z}[B](v_1, \dots, v_n) = 1$ iff $\zeta(v_1, \dots, v_n) = 1$ and $Z(v_1, \dots, v_n) > B$ is called *maximization constraint*.
- *A minimization or maximization constraint is also called an optimization constraint.*

The next definition couples optimization constraints and relaxations.

Definition 2.2 *Given an optimization constraint $\vartheta_{\zeta, Z}[B] : D(V_1) \times \dots \times D(V_n) \rightarrow \{0, 1\}$, let $\Delta := D(V_1) \times \dots \times D(V_n)$.*

- *We say that an optimization constraint $\vartheta_{\zeta, Z}[B]$ is consistent, iff for any given $1 \leq i \leq n$ and $v_i \in D(V_i)$, there exist $v_j \in D(V_j)$, $j \neq i$, such that $\vartheta_{\zeta, Z}[B](v_1, \dots, v_n) = 1$.*
- *Let $\vartheta_{\zeta, Z}[B]$ be a minimization constraint, and let $L : 2^\Delta \rightarrow \mathbb{R}$ such that for all $M_i \subseteq D(V_i)$, $1 \leq i \leq n$,*

$$L(M_1 \times \cdots \times M_n) \leq \min\{Z(v_1, \dots, v_n) \mid \zeta(v_1, \dots, v_n) = 1, v_i \in M_i, 1 \leq i \leq n\},$$

where $\min \emptyset = \infty$. We say that $\vartheta_{\zeta, Z}[B]$ is relaxed L -consistent, iff for any given $1 \leq i \leq n$ and $v_i \in D(V_i)$, $L(D(V_1) \times \cdots \times \{v_i\} \times \cdots \times D(V_n)) < B$.

- Analogously, let $\vartheta_{\zeta, Z}[B]$ be a maximization constraint, and let $U : 2^\Delta \rightarrow \mathbb{R}$ such that for all $M_i \subseteq D(V_i)$, $1 \leq i \leq n$,

$$U(M_1 \times \cdots \times M_n) \geq \max\{Z(v_1, \dots, v_n) \mid \zeta(v_1, \dots, v_n) = 1, v_i \in M_i, 1 \leq i \leq n\},$$

where $\max \emptyset = -\infty$. We say that $\vartheta_{\zeta, Z}[B]$ is relaxed U -consistent, iff for any given $1 \leq i \leq n$ and $v_i \in D(V_i)$, $U(D(V_1) \times \cdots \times \{v_i\} \times \cdots \times D(V_n)) > B$.

2.2 Variable Fixing for the Constrained Knapsack Problem

In a canonical IP formulation of the knapsack problem, there is one variable X_i for each item $i \in \{1, \dots, n\}$. The domain of each variable is defined as $D(X_i) := \mathbb{B}$. Further, the knapsack constraint is modeled by a function $\omega : \mathbb{B}^n \rightarrow \mathbb{B}$ with $\omega(x) = \omega(x_1, \dots, x_n) = 1$ iff $w^T x \leq C$. Finally, the objective function is $P : \mathbb{B}^n \rightarrow \mathbb{R}$ with $P(x) = P(x_1, \dots, x_n) := p^T x$. Given any lower bound $B \geq 0$, we consider the maximization constraint $\vartheta_{\omega, P}[B]$. Items of the CKP fall into either one of the following classes:

- items i that can be excluded from further investigation as they cannot be part of any improving solution, i.e.

$$P(x) \leq B \quad \forall x \in \{y \in F \mid y_i = 1\} \quad (1)$$

- items i that can be included into the knapsack as they must be part of any improving solution, i.e.

$$P(x) \leq B \quad \forall x \in \{y \in F \mid y_i = 0\} \quad (2)$$

- items that cannot be decided at the moment.

Propagation is expected to include or remove items that do not fall into the last class. Since showing that either (1) or (2) holds for an item i (i.e. to check the consistency of $\vartheta_{\omega, P}[B]$) generally requires to solve a CKP itself, complete propagation here is an NP-hard task. Therefore, we only check if the inequality holds for an upper bound U on $\text{CKP}[x_i = b]$, $b = 0$ or $b = 1$, i.e., if $U(\text{CKP}[x_i = b]) \leq B$.¹

In the KP literature, this idea has been used to reduce problem sizes initially or in selected nodes of a branch and bound tree. Especially when solving complex problems such as quadratic knapsack problems, variable fixing is of great importance [21]. In the next section, we give the definitions of some bounds that have been developed for the KP.

¹To improve the readability, here and in the following we write CKP or KP instead of \mathbb{B}^n , and identify $\text{CKP}[x_i = b]$ as well as $\text{KP}[x_i = b]$ with $\mathbb{B} \times \cdots \times \{b\} \times \cdots \times \mathbb{B}$, where $\{b\}$ is the i th factor.

2.3 Upper Bounds for Knapsack Problems

The effectiveness of a knapsack constraint propagation algorithm relies heavily on the quality of the bounds calculated. Following the presentation given in Chapter 2 of [17], we present some upper bounds that have been developed for the KP originally. They also apply to the CKP by relaxing it to a KP first. Obviously, ignoring all additional constraints often does not yield tight bounds on the objective. However, if the additional constraints satisfy certain properties, they can be incorporated in the objective function of a pure KP using Lagrangian relaxation. For additional linear constraints, there are ways of how this can be done effectively [9, 22]. Notice, that dropping all additional constraints allows to set $x_i := 0 \forall p_i \leq 0$ and $1 \leq i \leq n$. We therefore require all items to have positive profits.

Without loss of generality, we may assume that the items are ordered according to decreasing efficiency, i.e. $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}$. We define the *critical item* s of a knapsack problem as the first item that overloads the knapsack, that is $s = \min_j \{ \sum_{i=1}^j w_i > C \}$ (we omit the trivial case here where no such s exists). In [4], Dantzig showed that the linear relaxation of the 0-1 knapsack has the optimal value $\sum_{j=1}^{s-1} p_j + \bar{c} \frac{p_s}{w_s}$, where \bar{c} is defined as the remaining capacity of the knapsack after filling in the first $s-1$ items: $\bar{c} = C - \sum_{j=1}^{s-1} w_j$.

Let $\emptyset \neq M_1, \dots, M_n \subseteq \mathbb{B}$. Denote with $l_i := \min(M_i)$ the minimum, and with $r_i := \max(M_i)$ the maximum of M_i , $1 \leq i \leq n$. The first upper bound on KP is defined as $U_1 : 2^{\mathbb{B}^n} \rightarrow \mathbb{R}$ with

$$U_1(M_1 \times \dots \times M_n) := \max\{P(x_1, \dots, x_n) \mid \omega(x_1, \dots, x_n) = 1, x_i \in [l_i, r_i], 1 \leq i \leq n\}.$$

It holds, that

$$U_1 := U_1(KP) = \sum_{j=1}^{s-1} p_j + \left\lfloor \bar{c} \frac{p_s}{w_s} \right\rfloor. \quad (3)$$

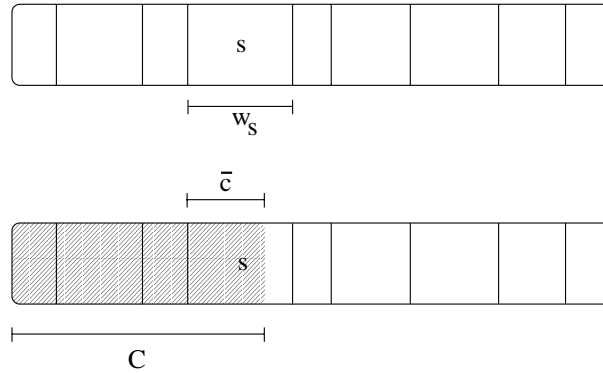


Figure 1: The width of each element is equivalent to its weight. The elements are ordered with respect to the efficiencies p_i/w_i . The leftmost element has the biggest efficiency, and the rightmost the smallest one. s marks the critical item in U_1 .

A second bound U_2 was introduced in [15]. It imposes the integrality of the critical item s . Either item s belongs to the optimal solution (leading to a value U^1) or not (leading to a

value U^0):

$$U^0 = \sum_{j=1}^{s-1} p_j + \left\lfloor \bar{c} \frac{p_{s+1}}{w_{s+1}} \right\rfloor. \quad (4)$$

$$U^1 = \sum_{j=1}^{s-1} p_j + \left\lfloor p_s - (w_s - \bar{c}) \frac{p_{s-1}}{w_{s-1}} \right\rfloor. \quad (5)$$

Defining U_2 as the maximum of U^0 and U^1 results in a bound dominating U_1 . Formally, let $\emptyset \neq M_1, \dots, M_n \subseteq \mathbb{B}$, and denote with s the critical item with respect to necessarily included and excluded items implicitly defined by the M_i . We set $U_2 : 2^{\mathbb{B}^n} \rightarrow \mathbb{R}$ with $U_2(\emptyset) := -\infty$, and

$$U_2(M_1 \times \dots \times M_n) := \max(U^0, U^1) - \sum_{i < s, M_i = \{0\}} p_i + \sum_{i > s, M_i = \{1\}} p_i.$$

It holds,

$$U_2 := U_2(KP) = \max(U^0, U^1) \leq U_1. \quad (6)$$

Instead of estimating the loss caused by the integrality of item s by the efficiency of the neighboring items of s , an even tighter bound can be obtained by calculating bounds U_1 on $KP[x_s = 0]$, and $KP[x_s = 1]$ [7, 10, 23]. Let $\bar{U}^0 := U_1(KP[x_s = 0])$, and $\bar{U}^1 := U_1(KP[x_s = 1])$. Then, $U_3 := \max(\bar{U}^0, \bar{U}^1)$ dominates U_1 and U_2 .

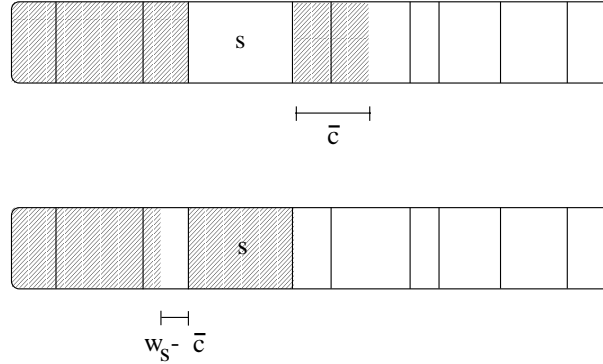


Figure 2: U_3 requires the integrality of item s . The figures show $U_1(KP[x_s = 0])$, and $U_1(KP[x_s = 1])$.

Fig. 1 and 2 give graphical interpretations of the bounds U_1 and U_3 . Obviously, all three bounds U_1, U_2, U_3 can be computed in time $O(n)$ after a preprocessing step of sorting the items according to decreasing efficiencies. This requires time $\Theta(n \log n)$. Balas and Zemel [1] developed an algorithm for the calculation of s using linear time without any preprocessing. But for the reduction algorithm we present in the following – just as in former reduction algorithms for the KP – the efficiency ordering is needed anyway. On top of that, we use an ordering of the items with respect to increasing weights.

In a tree search, both orderings can be calculated in an initial preprocessing step. After that, they can be reused in every search node. Within a column generation context, the weight ordering only has to be calculated once, but the efficiency ordering has to be recomputed every time new dual values of the master problem lead to a change of the objective in the successive CKPs.

2.4 Reduction Techniques for Knapsack Problems

A first reduction algorithm for KPs based on upper bound U_1 has been proposed by Ingargiola and Korsh in [12]. In a loop over all items $i = 1, \dots, n$, the algorithm determines $U_1(KP[x_i = b])$, $b \in \{0, 1\}$. Since each bound calculation takes linear time, the worst case complexity of this algorithm is $\Theta(n^2)$.

If bound U_2 is used instead of U_1 , more effective pruning can be achieved in the same asymptotic running time. In [16], Martello and Toth showed that the running time can be reduced to $O(n \log n)$ while keeping the solution quality of bound U_2 . The key idea of their algorithm is to compute the critical item s by binary search. We refer to the methods of Ingargiola and Korsh, and Martello and Toth as IKR, and MTR, respectively.

Dembo and Hammer [5] proposed a reduction algorithm (DHR) that runs in linear time $\Theta(n)$. They calculate the critical item s only once for the original problem. Within a loop they estimate the loss when removing/inserting item $i = 1, \dots, n$ by extrapolating the efficiency of item s , which allows to perform this step in constant time. As this extrapolation is less accurate than U_1 , their method is not as effective as IKR or MTR.

We do not know how to improve on the running time of reduction techniques based on the more efficient bounds U_1, U_2 , if the reduction algorithm is only called once. For such an application, the new method presented in the following and the one developed by Martello and Toth both use the same asymptotic time $\Theta(n \log n)$.

However, the situation changes if a reduction method is called many times for similar knapsack instances, as it is the case when applying a tree search. In CP e.g., constraints are triggered in every choice point. The subsequent instances only differ with respect to the sets of variables that have already been fixed. As we will see in the next section, such a situation allows to hide parts of the work in a preprocessing step that takes time $\Theta(n \log n)$. Provided with the information gathered in that preprocessing, every call to the reduction routine requires linear time only.

3 Fast Propagation Algorithms for CKPs

3.1 A Fast Propagation Algorithm based on Bound U_1 and U_2

We will now show how to reduce the running time of IKR and MTR to $\Theta(n)$ by making use of information generated in a preprocessing step requiring time $\Theta(n \log n)$. The bounds obtained are of the same quality as in the original algorithms. The key idea of the routine is to calculate the bounds of the reduced problems $U(\text{CKP}[x_j = b])$, $b \in \{0, 1\}$, in an order of increasing weight of the items j . Thereby, we obtain a sequence of critical items that is monotonically increasing. Thus, the critical item and the upper bound for the j th item (with respect to the weight ordering) can be transformed into the critical item and upper bound for the $(j + 1)$ th item by starting the calculation of $s(\text{KP}[x_{j+1} = b])$ at $s(\text{KP}[x_j = b])$.

The time consuming step in reduction algorithms using bound $U \in \{U_1, U_2\}$ is to determine the critical items $s(\text{KP}[x_i = b]) \forall 1 \leq i \leq n$, and $b \in \{0, 1\}$. Once these values are known, the calculation of the upper bounds and the reduction itself only require linear time. (In fact, in the following algorithm the bounds can be computed at the same time as the critical items. To clarify the argumentation, however, we just show how to calculate the latter.) By omitting the fractional parts, it is also possible to calculate lower bounds for the

KP. Reduction should only take place, after all lower bounds have been calculated [16]. For the CKP, however, the necessary feasibility checking makes the generation of lower bounds more complicated.

Although calculating $s(\text{KP}[x_i = b])$ for each single $i \in \{1, \dots, n\}$, $b \in \{0, 1\}$, generally takes linear time, the calculation of *all* these values also only requires time $\Theta(n)$ once we know an ordering $\sigma = (\sigma_1, \dots, \sigma_n)$ of the items according to their weight, i.e. $w_{\sigma_i} \leq w_{\sigma_j}$ iff $i \leq j$. The efficiency ordering of the items as well as the permutation σ can be obtained in a sorting step prior to any reduction and requiring time $\Theta(n \log n)$.

Given $s = s(\text{KP})$, we know that $U(\text{KP}[x_i = 1]) = U(\text{KP}) \forall i < s$, and $U(\text{KP}[x_i = 0]) = U(\text{KP}) \forall i > s$. Thus, we only need to calculate the arrays $S^1 := [s(\text{KP}[x_i = 1]) \mid i \geq s]$, and $S^0 := [s(\text{KP}[x_i = 0]) \mid i \leq s]$. We describe how to determine S^0 in the following. The calculation of S^1 is done analogously.

We iterate over all items $i < s$ in increasing order of weight. Like that, we can be sure that $s(\text{KP}[x_i = 0])$ increases monotonically with growing $i \in \{0, \dots, s - 1\}$. Thus, we can start the search for the next critical item at the position of the last one.

The following book keeping argument shows that this procedure only takes linear time. We estimate the computational effort of the reduction algorithm by assigning a unit cost (say, 1 €) to the items causing it:

- Every item $j \geq s$ that is being passed is charged 1 €. By “passed” we mean, that the item is being included entirely when iterating from one critical item to the other.
- Every item is charged 1 € each time it is being included fractionally.

The first group of items causes at most n € costs as the critical items are monotonically increasing: every item is being passed at most once. It remains the effort for all items that are being included fractionally. Obviously, there are at most as many fractionally included items as critical items. Therefore, this group of items also costs not more than n €. Thus, the costs for the entire computation are in $O(n)$.

Finally, the calculation of $s(\text{KP}[x_s = 0])$ can be performed in time that is linear in the number of items as well. Another possibility to calculate this value is to insert item s at the position corresponding to \bar{c} in the weight ordering of items and to calculate $s(\text{KP}[x_s = 0])$ just like the critical items for the exclusion of the other items.

Obviously, the above algorithm can be applied with bounds U_1 and U_2 . As a consequence, we have shown the following

Theorem 3.1 *After a $\Theta(n \log n)$ preprocessing step, relaxed U_2 -consistency for a knapsack constraint can be obtained in time $O(n)$ per choice point.*

It is easy to see that, for a constant number of choice points, MTR and the algorithm given above need the same running time of $\Theta(n \log n)$. If $\Omega(\log n)$ choice points have to be investigated, however, the time spent in the preprocessing is dominated by the accumulated time needed in the choice points. In that case, Theorem 3.1 implies

Corollary 3.2 *If $\Omega(\log n)$ search nodes are investigated, relaxed U_2 -consistency for a knapsack constraint can be obtained in amortized time $O(n)$ per choice point.*

Thus, in a typical CP search tree with $\Omega(\log n)$ search nodes, the method presented here is asymptotically optimal and superior to the algorithms proposed before.

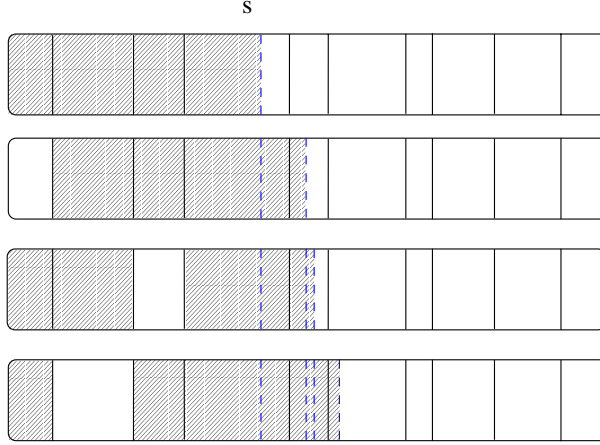


Figure 3: The figure illustrates the proceeding of the reduction algorithm presented for $KP[x_i = 0]$. The weight ordering in which the items are tested ensures that the critical item moves monotonically to the right.

3.2 More Effective Cost Based Filtering using Bound U_3

To strengthen the propagation abilities of the optimization constraint, we can use the stronger bound U_3 :

Let $1 \leq i \leq n$, $b \in \{0, 1\}$, $s_i^0 := s(KP[x_i = b, x_{s(KP[x_i=b])} = 0])$, and $s_i^1 := s(KP[x_i = b, x_{s(KP[x_i=b])} = 1])$. To calculate $U_3(KP[x_i = b])$, we need to compute s_i^0 , and s_i^1 . To do so, we first determine the values $s(KP[x_i = b])$ using the algorithm in Section 3.1. Then, we apply a binary search to determine s_i^0 and s_i^1 for all $1 \leq i \leq n$. This leads to a running time of $\Theta(n \log n)$. A similar idea has been introduced by Martello and Toth in [16].

Corollary 3.3 *Relaxed U_3 -consistency for a knapsack constraint can be obtained in time $O(n \log n)$ per choice point.*

For real life instances, using a binary search to determine the critical item of $KP[x_s = b_2, x_i = b_1]$ for $b_1, b_2 \in \{0, 1\}$, usually does not pay off as it is likely to be “close” to s . Thus, we consider this result to be of theoretical interest only. However, the algorithm above leads to another propagation algorithm that is asymptotically as efficient as the one presented in Section 3.1 (that runs in amortized linear time), but that is even more effective. In fact, the bound it uses to perform cost based filtering is at least as good as U_2 , but for some items it is even U_3 :

Let $1 \leq i \leq n$, $b \in \{0, 1\}$, $s := s(KP)$, $s_i^0 := s(KP[x_i = b, x_s = 0])$, and $s_i^1 := s(KP[x_i = b, x_s = 1])$. In contrast to the sequence of critical items as computed for U_3 , the second variable x_s that is being fixed remains the same for all s_i^0 , and s_i^1 . Again by using the algorithm in Section 3.1, we determine $U_2(KP[x_s = 0, x_i = b]) \forall 1 \leq i \leq n$, and then $U_2(KP[x_s = 1, x_i = b]) \forall 1 \leq i \leq n$. For any given $1 \leq i \leq n$, we check whether $\max\{U_2(KP[x_s = 0, x_i = b]), U_2(KP[x_s = 1, x_i = b])\} \leq B$. If so, we fix the value of x_i to $1 - b$.

It is easy to see that the bound calculated is at least as good as U_2 . For items $i < s$ with $s(KP[x_i = 0]) = s$ and items $i > s$ with $s(KP[x_i = 1]) = s$, however, propagation is even as

Name	see	Bound	preproc time	time per node
DHR	Sect. 2.4, [5]	$D/H - bound$	–	$\Theta(n)$
MTR	Sect. 2.4, [16]	U_2	$\Theta(n \log n)$	$\Theta(n \log n)$
lin U_1	Sect. 3.1	U_1	$\Theta(n \log n)$	$\Theta(n)$
lin U_2	Sect. 3.1	U_2	$\Theta(n \log n)$	$\Theta(n)$

Table 1: Characteristics of the four algorithms used in the experiments.

effective as for bound U_3 . Hence, we achieve an amortized linear time algorithm based on a ‘mix’ of U_2 and U_3 bounds.

4 Experiments

After we analyzed the new algorithm theoretically in Section 3.1, we now compare it numerically with different methods that were derived from KP reduction techniques presented in the literature. Our algorithms lin U_1 and lin U_2 are based on the amortized linear time reduction method described in Section 3.1, and use bounds U_1 and U_2 , respectively. Methods DHR, and MTR have been described in Section 2.4. We implemented all algorithms in the same CP environment. Table 1 summarizes the major characteristics for the candidates used in the experiments.

All experiments were run on a Sun Enterprise 450 Model 4300 (296 MHz) with 1 GB RAM, under Solaris 2.6. The reduction algorithms were implemented in C++ on top of Ilog Solver 5.0 [11].

4.1 Test Environment

To show the potential of the new propagation techniques, and to avoid cross talking, we omit all possible additional constraints and focus on the pure knapsack constraint. (For an example of a combination of the algorithms presented here and a shortest path constraint we refer to [22]). Accordingly, we also omit specially tailored tree search or branching strategies for pure KPs. Instead, we used the default settings of the underlying CP library.

A weak propagation algorithm, if started from scratch, will obviously need more choice points to find an optimal or near optimal solution of the problem than a good one. Therefore, to make the comparison fair, we initialize the lower bound with the optimal objective value $B \in \mathbb{R}$ and just measure the time and the number of choice points that each approach takes to prove optimality.

The generator code of David Pisinger [20] was used to produce random instances of two different classes of knapsack problems where the weights w_j are randomly distributed in $[1,1000]$, and the profits p_j are chosen as given below:

- *uncorrelated*: p_j randomly distributed in $[1,1000]$,
- *weakly correlated*: p_j randomly distributed in $[w_j - 100, w_j + 100]$, $p_j \geq 1$.

In all cases, the knapsack capacity is chosen as $C = \frac{1}{2} \sum_{j=1}^n w_j$. The problem sizes range from 10 to 20 000 items, and 100 knapsack problems were generated for each size and class.

We omit the classes of *strongly correlated* data ($p_j = w_j + 10$) and *subset-sum* data ($p_j = w_j$). It is known that the bounds described in Section 2.3 are not suited for these

Size n	uncorrelated		weakly correlated	
	cp	$time$	cp	$time$
10	37.77	0.01	73.74	0.01
20	1 455.80	0.16	28 736.07	2.91
30	141 338.82	15.50	16 771 406.92	1641.94
40	10 311 820.44	1410.07	—	—

Table 2: The pure CP approach for both problem classes. cp is the average number of choice points, $time$ the average time in seconds for 100 instances of the given size.

classes (which is easy to see as $\forall k : p_k/w_k \approx 1$). For them, bounds based on cardinality constraints have shown to be effective [14, 18]. In the application area that we focus on (see Section 1.2), however, it is justified to assume that the evolving KPs are more likely to fall into one of the classes we used for our tests.

4.2 Numerical Results

The simple approach for solving a CKP in a CP context would be to introduce a sum-constraint (i.e. $\sum_j w_j x_j \leq C$) plus a constraint stating that we are only looking for improving solutions (i.e. $\sum_j p_j x_j > B$). However, as shown in Table 2, that approach cannot compete at all with the other propagation methods. Both the number of choice points and the CPU time grow enormously when the problem size increases. A dash means that the average calculation for a test instance takes more than two hours. For both classes, only small problems with not more than 40 items can be solved within that time limit. The poor performance of the pure CP approach shows the need for sophisticated filtering techniques when knapsack constraints occur in a CP model. As will be shown in the following, more elaborate techniques are able to tackle problems of several 1000 items in a few seconds, generating only relatively few choice points.

4.2.1 Small Instances

Tables 3 and 4 show the average results of 100 different instances of the same data size n . We present the running time in seconds, and the number of choice points cp that the method visits. Table 5 shows a comparison of the different methods regarding the time per choice point for uncorrelated and weakly correlated data.

The Dembo/Hammer based propagation method needs to visit the largest amount of choice points among the four propagation algorithms. This matches the expected behaviour of a method that prunes with respect to weaker bounds. Due to the short time per choice point, though, it is only slightly slower than the other methods on uncorrelated data. Thus, the numerical results reflect the expected trade-off between an effective pruning and the time needed for it. In the presence of additional constraints (causing higher times spent per choice point that is needed for propagation), it is likely that a smaller number of choice points will result in a faster overall computation. $\text{lin}U_1$ uses fewer choice points than DHR, but is not as effective as the U_2 based algorithms, MTR and $\text{lin}U_2$. For the big instances, these two only visit between 50% and 65.6% of the choice points needed by DHR.

For weakly correlated data, $\text{lin}U_2$ only visits at most 69.7% of the choice points of the

Size <i>n</i>	DHR		linU ₁		linU ₂		MTR	
	<i>cp</i>	<i>time</i>	<i>cp</i>	<i>time</i>	<i>cp</i>	<i>time</i>	<i>cp</i>	<i>time</i>
10	2.43	0.00	0.87	0.00	0.67	0.00	0.67	0.00
20	5.47	0.00	2.68	0.00	2.35	0.00	2.35	0.00
40	7.20	0.00	3.61	0.00	3.22	0.00	3.22	0.00
60	10.18	0.00	6.07	0.00	5.26	0.00	5.26	0.00
80	13.96	0.01	8.43	0.00	7.04	0.00	7.04	0.00
100	14.21	0.01	8.20	0.00	6.75	0.00	6.75	0.00
200	24.85	0.02	17.16	0.02	14.47	0.01	14.47	0.01
300	32.47	0.04	22.57	0.03	18.76	0.02	18.76	0.02
400	38.19	0.05	27.69	0.04	23.28	0.04	23.28	0.04
500	46.50	0.08	33.64	0.06	28.68	0.05	28.68	0.05
600	63.61	0.11	48.67	0.09	40.95	0.08	40.95	0.08
700	54.67	0.11	41.16	0.09	34.53	0.08	34.53	0.08
800	69.92	0.16	51.76	0.13	42.38	0.11	42.38	0.11
900	68.89	0.17	51.76	0.14	42.35	0.13	42.35	0.12
1000	97.83	0.26	72.38	0.21	59.73	0.17	59.73	0.18

Table 3: Uncorrelated data instances. We give the average numbers for 100 test sets per size. *time* is the time in seconds, *cp* the number of choice points.

Size <i>n</i>	DHR		linU ₁		linU ₂		MTR	
	<i>cp</i>	<i>time</i>	<i>cp</i>	<i>time</i>	<i>cp</i>	<i>time</i>	<i>cp</i>	<i>time</i>
10	10.42	0.00	6.31	0.00	5.42	0.00	5.42	0.00
20	20.41	0.00	13.82	0.00	11.35	0.00	11.35	0.00
40	33.26	0.01	23.42	0.01	19.87	0.01	19.87	0.00
60	37.69	0.01	26.69	0.01	22.52	0.01	22.52	0.01
80	56.07	0.02	40.10	0.01	33.21	0.01	33.21	0.01
100	61.60	0.02	45.49	0.02	37.94	0.02	37.94	0.02
200	103.85	0.06	77.05	0.05	64.33	0.05	64.33	0.04
300	162.20	0.13	123.11	0.11	99.67	0.10	99.67	0.09
400	202.23	0.21	151.50	0.17	118.71	0.15	118.71	0.14
500	226.36	0.29	161.80	0.23	122.57	0.19	122.57	0.18
600	286.40	0.42	207.56	0.33	158.92	0.27	158.92	0.26
700	345.28	0.58	252.25	0.45	185.42	0.36	185.42	0.35
800	314.00	0.61	214.64	0.44	151.34	0.34	151.34	0.33
900	428.16	0.89	300.34	0.67	210.06	0.51	210.06	0.49
1000	451.74	1.04	313.50	0.78	220.33	0.60	220.33	0.57

Table 4: Weakly correlated data instances. We give the average numbers for 100 test sets per size. *time* is the time in seconds, *cp* the number of choice points.

Size	Type	DHR	linU₁	linU₂	MTR
<i>n</i>		time per cp	time per cp	time per cp	time per cp
500	uncorrelated	1.72	1.78	1.74	1.74
500	correlated	1.28	1.42	1.55	1.47
1000	uncorrelated	2.66	2.90	2.85	3.01
1000	correlated	2.30	2.49	2.72	2.59

Table 5: Uncorrelated and weakly correlated data instances. We give the average time per choice point in milliseconds for 100 test sets per size.

Size	DHR		linU₁		linU₂		MTR	
<i>n</i>	<i>cp</i>	<i>time</i>	<i>cp</i>	<i>time</i>	<i>cp</i>	<i>time</i>	<i>cp</i>	<i>time</i>
1000	97.83	0.26	72.38	0.21	59.73	0.17	59.73	0.18
2000	161.48	0.79	120.64	0.65	100.38	0.51	100.38	0.56
3000	202.34	1.59	148.43	1.31	118.90	1.00	118.90	1.06
4000	291.00	3.17	205.16	2.43	146.58	1.73	146.58	1.82
5000	360.47	4.82	245.32	3.79	184.83	2.65	184.83	2.98
6000	534.61	9.46	376.69	7.81	197.43	3.84	197.43	4.30
7000	620.48	12.90	431.55	10.11	294.18	6.78	294.18	7.57
8000	823.34	21.08	567.43	16.47	285.22	8.19	285.22	9.23
9000	1051.72	31.76	712.51	23.74	435.65	14.50	435.65	15.46
10000	1143.54	38.39	797.58	30.21	620.35	22.71	620.35	24.99

Table 6: Uncorrelated data. Comparison of running times for the new amortized linear time propagation algorithms and implementations of DHR, and MTR. We give the average time in seconds as well as the number of choice points for 100 test sets per size.

DHR routine. Moreover, $\text{lin}U_2$ slightly outperforms DHR with respect to the total running time. Notice, that the time per choice point spent by $\text{lin}U_2$ for weakly correlated instances is smaller than for uncorrelated data. The reason for this is, that the preprocessing time for initializing the more complex data structures for $\text{lin}U_2$, and for sorting the items according to weight and efficiency is spread over a much higher amount of choice points.

4.2.2 Big Instances

To get a clearer insight into the characteristics of the different algorithms, we performed some tests on bigger instances. Going up to 10 000 items, the disadvantages of the poor bounds used by $\text{lin}U_1$ and especially DHR become obvious. Due to a much higher amount of choice points, the total running times exceed those of $\text{lin}U_2$ and MTR (see Table 6).

Still, MTR and $\text{lin}U_2$ need about the same time on average. We assume that for smaller test instances, the binary search performed by MTR is faster than the more complex book keeping of $\text{lin}U_2$. As the problem size increases, however, the difference in efficiency becomes more noticeable, and $\text{lin}U_2$ outperforms MTR (see Tables 7 and 8).

A drawback of the new methods is the need for an initial sorting step in the preprocessing in which a profit and a weight ordering of all items are calculated. However, timing experiments show that this initial step costs about 0.06 seconds for 10 000 items and takes less than

Size	linU_2	MTR
n	time per cp	time per cp
500	1.74	1.74
1000	2.85	3.01
2000	5.08	5.58
4000	11.80	12.42
8000	28.71	32.36
16000	71.71	75.42

Table 7: Uncorrelated data. Comparison of running times per choice point for the new amortized linear time propagation algorithm based on bound U_2 and the implementation of MTR. We give the average time per choice point in milliseconds for 100 test sets per size.

Size	uncorrelated			weakly correlated		
	n	cp	linU_2 time	MTR time	cp	linU_2 time
10 000	620.35	22.71	24.99	1626.78	60.98	66.58
11 000	629.43	26.38	28.76	2572.45	110.47	121.08
12 000	604.87	28.04	32.31	2590.45	125.40	137.21
13 000	1341.42	69.30	77.31	2694.07	142.13	156.26
14 000	875.71	50.42	56.96	3520.18	206.68	228.54
15 000	1041.80	64.60	70.74	2818.97	185.33	204.80
16 000	1256.73	90.12	94.78	2164.99	154.56	172.14
17 000	1670.81	124.53	139.63	3145.36	250.59	276.93
18 000	2580.28	205.81	227.81			
19 000	2870.68	243.05	274.93			
20 000	2750.36	256.88	288.15			

Table 8: Comparison of running times of $\text{lin}U_2$ and MTR on uncorrelated and weakly correlated data. cp is the number of choice points, $time$ the running time in seconds.

0.01 seconds for 1000 items. According to Table 6, the total running time for these problem sizes is much higher. Hence, the preprocessing time can be neglected in praxis.

5 Conclusions

We proposed a formal definition of optimization constraints and relaxed L/U -consistency. Propagation based on these concepts has proven to be quite successful in recent years. Based on relaxation bounds for KP, we introduced a new reduction algorithm that runs in amortized time $\Theta(n)$ for $\Omega(\log n)$ calls. This algorithm can be used efficiently as propagation routine when solving the CKP in a CP context.

In a CP search, the efficiency of the algorithm developed depends on the number of choice points and the time needed per choice point: The more choice points are investigated during the search, the less dominant are the preprocessing times for initialization and sorting. And if more time per choice point is spent by other routines – that propagate additional constraints

of the CKP or calculate expensive bounds on the objective – the more important is an effective pruning behaviour that justifies a higher effort spent per choice point.

Experiments show that in a tree search the algorithm is as effective as another method based on a reduction technique previously proposed by Martello and Toth for KP. However, theoretical analysis and numerical comparison show, that the new method is asymptotically more efficient. Finally, the routines presented have already been used successfully in a Lagrangian relaxation based approach for a multimedia application.

References

- [1] E. Balas and E. Zemel. An algorithm for large-scale zero-one knapsack problems. *Operations Research*, 28:119–148, 1980.
- [2] A. Caprara, D. Pisinger, P. Toth. Exact solution of the Quadratic Knapsack Problem. *INFORMS Journal on Computing*, 11:125–137, 1999.
- [3] C. Chu and J. Antonio. Approximation algorithm to solve real-life multicriteria cutting stock problems. *Operations Research*, 47(4):495–508, 1999.
- [4] G.B. Dantzig. Discrete variable extremum problems. *Operations Research*, 5:266–277, 1957.
- [5] R.S. Dembo and P.L. Hammer. A reduction algorithm for knapsack problems. *Methods of Operations Research*, 36:49–60, 1980.
- [6] T. Fahle, U. Junker, S.E. Karisch, N. Kohl, M. Sellmann, B. Vaaben. Constraint programming based column generation for crew assignment. To appear in *Journal of Heuristics*.
- [7] D. Fayard and G. Plateau. An algorithm for the solution of the 0-1 knapsack problem. *Computing*, 28:269–287, 1983.
- [8] F. Focacci, A. Lodi, M. Milano. Cost-Based Domain Filtering. *Proceedings of the CP'99* Springer LNCS 1713:189–203, 1999.
- [9] F. Focacci, A. Lodi, M. Milano. Cutting Planes in Constraint Programming: An Hybrid Approach. *Proceedings of the CP-AI-OR'00*, Paderborn Center for Parallel Computing, Technical Report tr-001-2000:45–51, 2000.
- [10] P.D. Hudson. Improving the branch and bound algorithm for the knapsack problem. Queen's University Research Report, Belfast, 1977.
- [11] ILOG. ILOG SOLVER. Reference manual and user manual. V5.0, ILOG, 2000.
- [12] G.P. Ingargiola and J.F. Korsh. A reduction algorithm for zero-one single knapsack problems. *Management Science*, 20:460–463, 1973.
- [13] U. Junker, S.E. Karisch, N. Kohl, B. Vaaben, T. Fahle, M. Sellmann. A Framework for Constraint programming based column generation. *Proceedings of the CP'99* Springer LNCS 1713:261–274, 1999.

- [14] S. Martello, D. Pisinger, P. Toth. Dynamic programming and tight bounds for the 0-1 knapsack problem. *Management Science*, 45:414–424, 1999.
- [15] S. Martello and P. Toth. An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research*, 1:169–175, 1977.
- [16] S. Martello and P. Toth. A new algorithm for the 0-1 knapsack problem. *Management Science*, 34:633–644, 1988.
- [17] S. Martello and P. Toth. Knapsack Problems – Algorithms and Computer Implementations. *Wiley Interscience*, 1990.
- [18] S. Martello and P. Toth. Upper Bounds and Algorithms for hard 0-1 knapsack problems. *Operations Research*, 45(5):768–778, 1997.
- [19] G. Ottosson and E.S. Thorsteinsson. Linear Relaxation and Reduced-Cost Based Propagation of Continuous Variable Subscripts. *Proceedings of the CP-AI-OR'00*, Paderborn Center for Parallel Computing, Technical Report tr-001-2000:129–138, 2000.
- [20] D. Pisinger. An expanding-core algorithm for the exact 0-1 knapsack problem. *European Journal of Operational Research*, 87:175–187, 1995.
- [21] O. Porto, M. de Moraes, A. Lucena. A relax and cut algorithm for the quadratic knapsack problem. *Proceedings of the ISMP'00*, 17th International Symposium on Mathematical Programming, Atlanta, 2000.
- [22] M. Sellmann and T. Fahle. CP-based Lagrangian Relaxation for a Multimedia Application. Technical Report, University of Paderborn, 2001.
- [23] P.R.C. Vilela and C.T. Bornstein. An improved bound for the 0-1 knapsack problem. Report ES31-83, COPPE-Federal University of Rio de Janeiro, 1983.