Cost based Filtering for the Constrained Knapsack Problem *

Torsten Fahle and Meinolf Sellmann ({tef,sello}@uni-paderborn.de)
University of Paderborn
Department of Mathematics and Computer Science
Fürstenallee 11
D-33102 Paderborn

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, cost based filtering, optimization constraints, reduction algorithms

* 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 the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

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 (Focacci et al., 1999). Optimization constraints are used for pruning and to include (exclude) items that must (cannot) be part of any improving solution. We introduce propagation algorithms to perform pruning and cost based filtering for Constrained Knapsack Problems (CKPs).

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 the presence of additional constraints that have to be propagated in addition to the optimization constraint, 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 domain reduction. On the other hand, fast KP reduction algorithms using weak bounds, such as the algorithm developed by Dembo and Hammer (1980), are not effective enough for more complex CKPs.

Based on reduction techniques for KP, we develop propagation routines for knapsack constraints. We present several new propagation 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 (1977). 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.

Recently, Trick proposed a dynamic programming approach for propagation of knapsack constraints (Trick, 2001). We compare that approach with the one presented here in section 1.3.

Junker et al. (1999) developed a framework for the integration of CP and OR within column generation approaches, the so called Constraint Based Column Generation. 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 applied to the Crew Assignment Problem, where the subproblem is a Constrained Shortest Path Problem (Fahle et al., 2002).

Another important class of subproblems that evolve when following a column generation approach are (Constrained) Knapsack Problems. They evolve for example when solving a (Constrained) Cutting Stock problem. To motivate the work presented, we show exemplary how CKPs can be used when generating columns for that problem.

1.1. CONSTRAINED KNAPSACK PROBLEMS

The CKP is 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.

Let $C, n, w_1, \ldots, w_n \in \mathbb{N}$; $p_1, \ldots, p_n \in \mathbb{Z}$. *$C$ 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* $w := (w_1, \ldots, w_n)^T$, *and* $p := (p_1, \ldots, p_n)^T$.

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* $R := \{r_1, \ldots, r_k \mid r_j : \mathbb{B}^n \to \mathbb{B} \quad \forall\, 1 \leq j \leq k\}$. *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* $r(x) = 1\ \forall\ r \in R$. $F(R) := \{x \in G \mid x \textit{ is feasible}\}$ *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 *is then to*

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

Notice, that for the unconstrained KP it holds, $F = G$. Here, we investigate the general case of $F \subseteq G$. Generally, 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 (Martello and Toth, 1990) 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 $i \in \{1, \ldots, n\}$ is either included or excluded in any feasible solution. We refer to those problems with CKP$[x_i = 1]$ or 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 (Sellmann and Fahle, 2001a), 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 (Junker et al., 1999) 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 (Chu and Antonio, 1999)) 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. (2000) 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 (Caprara et al., 1999). The algorithms proposed in Section 3 may help to increase the performance of the overall approach in (Porto et al., 2000).

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 (Martello et al., 1999; Pisinger, 1995). For these algorithms it is not straightforward 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 are likely to fail. This motivates the decision to make use of efficiency orderings of the knapsack items 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.

Trick (2001) derives a hyper-arc-consistency approach from some dynamic programming method designed for pure KPs. The problems considered in that paper differ from the ones described in this paper: the input is a *two sided knapsack*, i.e. a linear constraint of the form $L \leq \sum_i W_i x_i \leq C$. The running time depends heavily on C and L in practice, and is theoretically bounded in $O(nC^2)$ only. That is, the algorithm is pseudo-polynomial in C. Also, the space requirement of $O(nC)$ depends on C. As an advantage, the approach works independently of the type of the objective function.

In contrast, the approach presented here considers *one sided knapsacks* with $\sum_i W_i x_i \leq C$, runs in amortized linear time, uses $O(n)$ space, and is tailored to linear cost functions only.

1.4. Outline of the Paper

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 (Focacci et al., 1999; Focacci et al., 2000; Junker et al., 1999; Fahle et al., 2002; Ottosson and Thorsteinsson, 2000).

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

DEFINITION 2. *Let $B \in \mathbb{R}$ denote an upper/lower bound on the objective Z to be minimized/maximized.*

- $\vartheta_{\zeta,Z}[B] : D(V_1) \times \cdots \times D(V_n) \to \{0,1\}$ *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.
- $\vartheta_{\zeta,Z}[B] : D(V_1) \times \cdots \times D(V_n) \to \{0,1\}$ *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 3. *Given an optimization constraint $\vartheta_{\zeta,Z}[B] : D(V_1) \times \cdots \times D(V_n) \to \{0,1\}$, let $\Delta := D(V_1) \times \cdots \times D(V_n)$. Further, denote with 2^Δ the set of all subsets of Δ.*

- *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 $L : 2^\Delta \to \mathbb{R}$ 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* $U : 2^\Delta \to \mathbb{R}$ *such that for all* $M_i \subseteq D(V_i)$, $1 \leq i \leq n$,

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

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* $v_i \in D(V_i)$, $U(D(V_1) \times \cdots \times \{v_i\} \times \cdots \times D(V_n)) > B$.

When solving an optimization problem, B is used as a no-good and is usually determined as the value of the incumbent solution. As the quality of B is crucial for the effectiveness of the propagation algorithm, in practice a primal heuristic is often applied to determine a fairly good solution prior to the tree search.

From the definition, relaxed L-consistency (relaxed U-consistency follows analogously) can the easier be achieved the weaker L is. For $L \equiv -\infty$, any propagation algorithm has nothing to do, whereas the tightest "relaxation" is achieved when $L(M_1 \times \cdots \times M_n) = \min\{Z(v_1, \ldots, v_n) \mid \zeta(v_1, \ldots, v_n) = 1,\ v_i \in M_i,\ 1 \leq i \leq n\}$. That is, the choice of L determines the degree of propagation. Usually, L is chosen as a tight bound that can be computed quickly.

Clearly, optimization constraints are closely related to global constraints and generalized arc-consistency (e.g. Régin, 1994) as they link a (global) constraint together with the restriction to improve on the objective function. The main contribution here consists in the definition of relaxed consistency that has been widely used in the OR community before to prune in the search tree. The idea is similar to the definition of bound consistency that can also be achieved more easily than general arc consistency, and that has proven valuable for many applications.

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 $i \in \{1, \ldots, n\}$. The domain of each variable is defined as $D(x_i) := \mathbb{B}$. Further, the knapsack constraint is modeled by a function $\omega : \mathbb{B}^n \to \mathbb{B}$ with $\omega(x) = \omega(x_1, \ldots, x_n) = 1$ iff $w^T x \leq C$. Finally, the objective function is $P : \mathbb{B}^n \to \mathbb{R}$ with $P(x) = P(x_1, \ldots, x_n) := p^T x$. 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.

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

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

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

- 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 satisfiability 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 CKP$[x_i = b]$, $b = 0$ or $b = 1$, i.e., if $U(\mathbb{B} \times \cdots \times \{b\} \times \cdots \times \mathbb{B}) \leq B$. Then, we write $U(\text{CKP}[x_i = b]) \leq B$.[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 (Porto et al., 2000). In the next section, we give the definitions of some bounds that have been developed for the KP.

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 Martello and Toth (1990), we present some upper bounds that have been originally developed for the maximization problem KP. 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 (Focacci et al., 2000; Sellmann and Fahle, 2001a). 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. $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}$. We define the *critical item* s of a knapsack problem as the first item that overloads the

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

knapsack, that is $s = \min_j\{\sum_{i=1}^{j} w_i > C\}$ (we omit the trivial case here where no such s exists). Dantzig (1957) showed that the linear relaxation of the 0-1 knapsack has the optimal value $\sum_{j=1}^{s-1} p_j + \overline{c}\frac{p_s}{w_s}$, where $\overline{c}$ is defined as the remaining capacity of the knapsack after filling in the first $s-1$ items: $\overline{c} = C - \sum_{j=1}^{s-1} w_j$.

Let $\emptyset \neq M_1, \ldots, M_n \subseteq \mathbb{B}$. 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 $U_1 : 2^{\mathbb{B}^n} \to \mathbb{R}$ with

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

It holds,

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

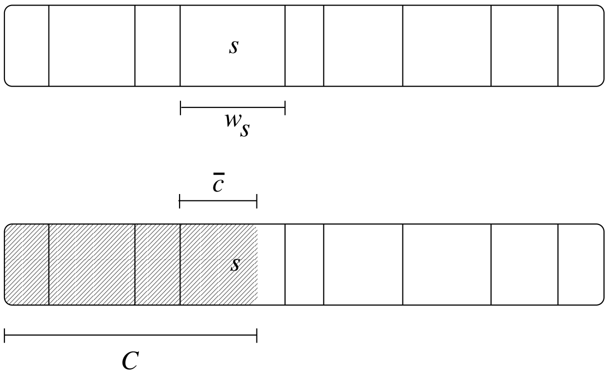

Figure 1. The width of each element is equivalent to its weight. The elements are ordered with respect to the efficiencies p_i/w_i. The leftmost element has the biggest efficiency, and the rightmost the smallest one. s marks the critical item in U_1.

A second bound U_2 was introduced in (Martello and Toth, 1977). 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):

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

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

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

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

It holds,

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

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 $\mathrm{KP}[x_s = 0]$, and $\mathrm{KP}[x_s = 1]$ (Fayard and Plateau, 1983; Hudson, 1977; Villela and Bornstein, 1983). Let $\overline{U}^0 := U_1(\mathrm{KP}[x_s = 0])$, and $\overline{U}^1 := U_1(\mathrm{KP}[x_s = 1])$. Then, $U_3 := \max(\overline{U}^0, \overline{U}^1)$ dominates U_1 and U_2. An even tighter bound could be obtained by using U_2 instead of U_1 in the definition of $\overline{U}^0$ and $\overline{U}^1$ and so on.

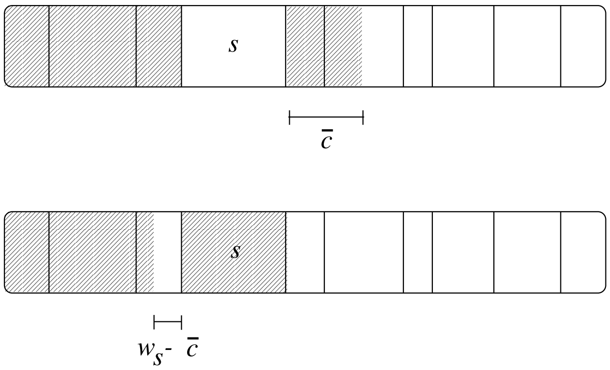

Figure 2. U_3 requires the integrality of item s. The figures show $U_1(\mathrm{KP}[x_s = 0])$, and $U_1(\mathrm{KP}[x_s = 1])$.

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 (1980) developed an algorithm for the calculation of s using linear time without any preprocessing. But for the reduction algorithm that 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 (1973). 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. Martello and Toth (1988), 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 (1980) 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/including 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.

Though being developed a decade or more ago, DHR or MTR are still vital ingredients in state-of-the-art solvers for pure KP or quadratic KP (see e.g. Pisinger, 1995, Pisinger, 1999, Porto et al., 2000).

The algorithm we present in the following does not 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 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. Within a tree search, we try to prune the search or to apply domain filtering after every branching step. 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. Again, let (C)KP$[x_j = b]$ denote $\mathbb{B} \times \cdots \times \{b\} \times \cdots \times \mathbb{B}$, $b \in \{0,1\}$, and $s(M_1 \times \cdots \times M_n) = \min_j\{\sum_{i \le j \mid M_i = \mathbb{B}} w_i > C - \sum_{i \mid M_i = \{1\}} w_i\}$ denote the critical item of (C)KP$[x_j = b]$. The key idea of the routine is to calculate the bounds of the reduced problems $U(\mathrm{CKP}[x_j = b])$ 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 jth 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 $s(\mathrm{KP}[x_{j+1} = b])$ at $s(\mathrm{KP}[x_j = b])$.

The time consuming step in reduction algorithms using bound U_1, U_2, resp., is to determine the critical items $s(\mathrm{KP}[x_i = b])\ \forall\ 1 \le i \le n$, 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 (Martello and Toth, 1988). For the CKP, however, the necessary feasibility checking with respect to additional constraints makes the generation of lower bounds more complicated.

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

Given $s = s(\mathrm{KP})$, we know that $U(\mathrm{KP}[x_i = 1]) = U(\mathrm{KP})\ \forall\ i < s$, and $U(\mathrm{KP}[x_i = 0]) = U(\mathrm{KP})\ \forall\ i > s$. Thus, we only need to calculate the arrays $S^1 := [s(\mathrm{KP}[x_i = 1]) \mid i \ge s]$, and $S^0 := [s(\mathrm{KP}[x_i = 0]) \mid i \le s]$. 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 $s(\mathrm{KP}[x_i = 0])$ increases monotonically with growing $i \in \{0,\ldots,s-1\}$. 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 $j \ge s$ 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 $s(\mathrm{KP}[x_s = 0])$ 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 $\overline{c}$ in the weight ordering of items and to calculate $s(\mathrm{KP}[x_s = 0])$ 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 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 1 implies

COROLLARY 1. *If propagation is triggered in $\Omega(\log n)$ search nodes,* 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.

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:

$U_3(KP)$ is obtained by calculating bound U_1 on $\mathrm{KP}[x_s = 0]$, and $\mathrm{KP}[x_s = 1]$. For propagation based on that bound, we need to compute s_i^b, $b \in \{0,1\}$, the critical items of those restricted KPs if additionally $x_i = b$: Let $1 \leq i \leq n$, $b \in \{0,1\}$. Then, $s_i^0 := s(\mathrm{KP}[x_i = b, x_{s(\mathrm{KP}[x_i=b])} = 0])$, and $s_i^1 := s(\mathrm{KP}[x_i = b, x_{s(\mathrm{KP}[x_i=b])} = 1])$.

To do so efficiently, we first determine the values $s(\mathrm{KP}[x_i = b])$ 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 (1988).

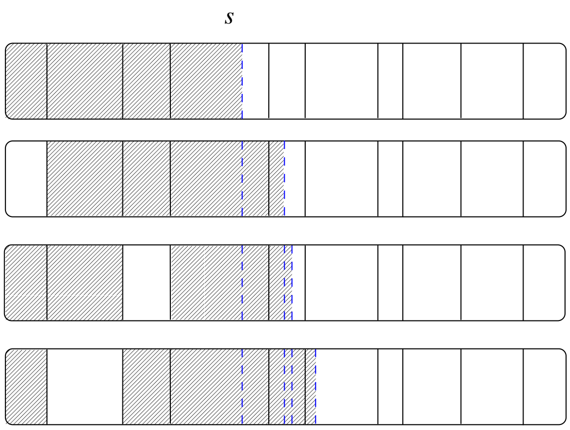

Figure 3. The figure illustrates the proceeding of the reduction algorithm presented for KP$[x_i = 0]$. The weight ordering in which the items are tested ensures that the critical item moves monotonically to the right.

COROLLARY 2. *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 $1 \leq i \leq n$, $b \in \{0,1\}$, $s := s(KP)$, $s_i^0 := s(\text{KP}[x_i = b, x_s = 0])$, and $s_i^1 := s(\text{KP}[x_i = b, x_s = 1])$. In contrast to the sequence of critical items that is 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 $U_2(\text{KP}[x_s = 0, x_i = b])\ \forall\ 1 \leq i \leq n$, and then $U_2(\text{KP}[x_s = 1, x_i = b])\ \forall\ 1 \leq i \leq n$. For any given $1 \leq i \leq n$, we check whether $\max\{U_2(\text{KP}[x_s = 0, x_i = b]), U_2(\text{KP}[x_s = 1, x_i = b])\} \leq B$. 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 $s(\text{KP}[x_i = 0]) = s$ and items $i > s$ with $s(\text{KP}[x_i = 1]) = s$, however, propagation is even as 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.

3.3. Cost Based Filtering for Special Constrained Knapsack Problems

Before we evaluate the propagation algorithms empirically, we would like to discuss their applicability to two special variants of the (constrained) knapsack problem that have been introduced in the literature.

3.3.1. *Multi-Dimensional Knapsack Problems*

The multi-dimensional knapsack problem consists in the maximization of a given profit function with respect to two or more given capacity constraints. The problem can be viewed as a collection of m pure knapsack problems sharing one objective:

$$\begin{array}{ll} \max & \sum_j p_j x_j \\ s.t. & \sum_j w_{i,j} x_j \ \leq \ C_i, \ \ i = 1, \ldots, m \\ & \quad x_j \in \{0,1\} \end{array} \tag{7}$$

Thus, for each of the capacity constraints we can define an optimization constraint and perform cost based filtering using the propagation algorithms we just presented. This approach, however, suffers a setback from the fact, that the bounds computed in each optimization constraint ignore all constraints except one. Therefore, the bounds are not tight, and filtering is less effective than it could and should be.

In (Sellmann and Fahle, 2001a), we develop of generic method for the coupling of linear optimization constraints to one global optimization constraint, the *CP-based Lagrangian Relaxation*. When applied to multi-dimensional knapsack problems, the filtering algorithm for the composite constraint uses the propagation routines of the individual knapsack constraints incorporating the other constraints in a Lagrangian objective. We have shown that this approach is clearly favorable compared to the loose connection of optimization constraints that interact by domain reduction only.

Note, however, that the asymptotic complexity improvements we introduce in this paper are lost when applying the knapsack filtering algorithm in the context of CP-based Lagrangian relaxation, because for each Lagrangian subproblem the objective changes. Thus, the efficiency ordering has to be recomputed which then dominates the algorithmic complexity. Noteworthy, that problem does not occur when the filtering algorithms presented here are applied to column generation subproblems (as in CP-based column generation), because the objective remains fixed for the entire tree search that is applied to compute a new column. Thus, the efficiency ordering of the knapsack items has to be recomputed only when a new subproblem is set up.

3.3.2. *Bounded Knapsack Problems*
Bounded knapsack problems generalize the 0-1 KP by defining individual bounds on the solution vector:

$$\begin{array}{ll} \max & \sum_j p_j x_j \\ s.t. & \sum_j w_j x_j \leq C \\ & x_j \in \{0,1,2,\ldots,u_j\} \end{array} \tag{8}$$

Obviously, (8) can be transformed into a CKP by replacing one original x_j by u_j new variables $x'_{j,k} \in \{0,1\}, k = 1,\ldots,u_j$. (Notice, that a finite u_j always exists, as $x_j \leq \lfloor \frac{C}{w_j} \rfloor$.). Then the algorithms presented before could be applied. That approach, however, artificially enlarges the number of variables and ignores the additional structure of (8) completely.

We can do better by extending U_1 and U_2 to general integer bounds for KP. That is, we chose the critical item as $s := \min_j \{\sum_{i=1}^{j} u_i \cdot p_i > C\}$. Then U_1 can be re-written as $U_1(KP) = \sum_{j=1}^{s-1} u_j \cdot p_j + \left\lfloor \overline{c} \frac{p_s}{w_s} \right\rfloor$, where $\overline{c} = C - \sum_{j=1}^{s-1} u_j \cdot w_j$. For a detailed discussion of such generalizations, and an extension of U_2, we refer to (Martello and Toth, 1990, pp. 84ff.). Taking these extended bounds, efficient propagation for the bounded knapsack problem then is easily achieved by the algorithms proposed in Sec. 3.1, 3.2.

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. 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 (Ilog, 2000).

4.1. TEST ENVIRONMENT

To show the potential of the new propagation techniques, and to avoid cross talking with other constraints, we decided to base the experiments on pure knapsack problems only. Like that, we get a clear view on the performance of each filtering algorithm without disturbing interferences that can evoke easily when using more complex settings incorporating additional constraints. (For an example of a combination of the algorithms presented here and a shortest path constraint we refer to (Sellmann and Fahle, 2001a and 2001b). 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 word of care is necessary here: even though our experiments are based on pure KP data, the filtering algorithms we developed are not suited for state-of-the-art KP solvers. Also, we do not claim that the solvers we implemented are competitive to the best KP solvers (see Section 1.3). Our focus here is clearly on constrained knapsack problems.

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 $B \in I\!R$ and just measure the time and the number of choice points that each approach takes to prove optimality.

The generator code of David Pisinger (Pisinger, 1995) 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 $[w_j - 100, w_j + 100] \cap [1, 1100]$

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 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 (Martello et al., 1999; Martello and Toth, 1997). 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. The Opponents

The algorithms referred to as linU_1 and linU_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 I summarizes the major characteristics for the candidates used in the experiments. All methods need $O(n)$ memory for the propagation stack and for the different orderings used. Within a choice point, only $O(1)$ memory is required additionally.

Notice, that in our experiments we do not evaluate the filtering algorithm based on a mixture of bound U_2 and U_3 that was sketched in Section 3.2. The propagation algorithm based on this mixed bound visits only slightly fewer choice points than linU_2, but therefore requires a much higher computation

Table I. Characteristics of the four algorithms used in the experiments.

Name	**see**	**Bound**	**preproc time**	**time per node**
DHR	Sect. 2.4,	$D/H - bound$	–	$\Theta(n)$
MTR	Sect. 2.4,	U_2	$\Theta(n \log n)$	$\Theta(n \log n)$
linU_1	Sect. 3.1	U_1	$\Theta(n \log n)$	$\Theta(n)$
linU_2	Sect. 3.1	U_2	$\Theta(n \log n)$	$\Theta(n)$

time. Remember from Section 3 that the work that has to be done to perform propagation using bound U_2 is almost the same as using bound U_1. But when using the mixed bound, the workload is twice as large as that for bound U_1.

As we will show in this Section, we are facing a trade off between the time needed per choice point and the reduction of choice points that can be achieved by using tighter bounds. Within the test environment that we have chosen for our experiments, a slight reduction of choice points does not justify a much higher effort undertaken in every choice point. Therefore, the propagation algorithm based on the mixed bound is of interest only in the context of a more complex CKP incorporating additional and possibly hard side constraints that would make even small reductions of choice points more favorable. However, in the KP setting we use here to avoid cross talking with additional constraints and to evaluate the pure performance of the different propagation algorithms, the algorithm developed in Section 3.2 is not competitive.

4.3. 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 II, that approach cannot compete at all with the other propagation methods. Both the number of choice points and the CPU time grow exponentially 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.

Table II. 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.

Size	**uncorrelated**		**weakly correlated**	
n	*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 III. 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	**DHR**		**lin**U_1		**lin**U_2		**MTR**	
n	*cp*	*time*	*cp*	*time*	*cp*	*time*	*cp*	*time*
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

4.3.1. *Small Instances*

Tables III and IV 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 V shows a comparison of the different methods regarding the time per choice point for uncorrelated and weakly correlated data.

Table IV. 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	**DHR**		**linU_1**		**linU_2**		**MTR**	
n	cp	$time$	cp	$time$	cp	$time$	cp	$time$
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

The Dembo/Hammer based propagation method needs to visit the largest amount of choice points among the four propagation algorithms tested. This matches the expected behavior 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. linU_1 uses fewer choice points than DHR, but is not as effective as the U_2 based algorithms, MTR and linU_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, linU_2 only visits at most 69.7% of the choice points of the DHR routine. Moreover, linU_2 slightly outperforms DHR with respect to the total running time. Notice, that the time per choice point spent by linU_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 linU_2, and for sorting the items according to weight and efficiency is spread over a much higher amount of choice points.

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

Size n	**Type**	**DHR** time/cp	**linU_1** time/cp	**linU_2** time/cp	**MTR** time/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 VI. 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.

Size	**DHR**		**linU_1**		**linU_2**		**MTR**	
n	cp	$time$	cp	$time$	cp	$time$	cp	$time$
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

4.3.2. *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 linU_1 and especially DHR become obvious. Due to a much higher amount of choice points, the total running times exceed those of linU_2 and MTR (see Table VI).

Still, MTR and linU_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 linU_2. As the problem size increases, however, the difference in efficiency becomes more noticeable, and linU_2 outperforms MTR (see Tables VII and VIII).

Table VII. 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 n	**lin**U_2 **(time per cp)**	**MTR (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 VIII. Comparison of running times of linU_2 and MTR on uncorrelated and weakly correlated data. *cp* is the number of choice points, *time* the running time in seconds.

	uncorrelated			**weakly correlated**		
Size		**lin**U_2	**MTR**		**lin**U_2	**MTR**
n	*cp*	*time*	*time*	*cp*	*time*	*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	2980.91	251.43	279.63
19 000	2870.68	243.05	274.93	4871.67	435.33	476.97
20 000	2750.36	256.88	288.15	4319.27	405.56	452.50

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

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 behavior 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 combination with other constraints in a Lagrangian relaxation based approach for a multimedia application (Sellmann and Fahle, 2001a).

Acknowledgements

We would like to thank two anonymous referees for their helpful comments.

References

E. Balas and E. Zemel. An algorithm for large-scale zero-one knapsack problems. *Operations Research*, 28:119–148, 1980.

A. Caprara, D. Pisinger, P. Toth. Exact solution of the Quadratic Knapsack Problem. *INFORMS Journal on Computing*, 11:125–137, 1999.

C. Chu and J. Antonio. Approximation algorithm to solve real-life multicriteria cutting stock problems. *Operations Research*, 47(4):495–508, 1999.

G.B. Dantzig. Discrete variable extremum problems. *Operations Research*, 5:266–277, 1957.

R.S. Dembo and P.L. Hammer. A reduction algorithm for knapsack problems. *Methods of Operations Research*, 36:49–60, 1980.

T. Fahle, U. Junker, S.E. Karisch, N. Kohl, M. Sellmann, B. Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8:59–81, 2002.

D. Fayard and G. Plateau. An algorithm for the solution of the 0-1 knapsack problem. *Computing*, 28:269–287, 1983.

F. Focacci, A. Lodi, M. Milano. Cost-Based Domain Filtering. *Proceedings of the CP'99* Springer LNCS 1713:189–203, 1999.

F. Focacci, A. Lodi, M. Milano. Cutting Planes in Constraint Programming: An Hybrid Approach. *Proceedings of the CP-AI-OR'00*, pp. 45–51, 2000.

P.D. Hudson. Improving the branch and bound algorithm for the knapsack problem. Queen's University Research Report, Belfast, 1977.

ILOG. ILOG SOLVER. Reference manual and user manual. V5.0, ILOG, 2000.

G.P. Ingargiola and J.F. Korsh. A reduction algorithm for zero-one single knapsack problems. *Management Science*, 20:460–463, 1973.

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.

S. Martello, D. Pisinger, P. Toth. Dynamic programming and tight bounds for the 0-1 knapsack problem. *Management Science*, 45:414–424, 1999.

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.

S. Martello and P. Toth. A new algorithm for the 0-1 knapsack problem. *Management Science*, 34:633–644, 1988.

S. Martello and P. Toth. Upper Bounds and Algorithms for hard 0-1 knapsack problems. *Operations Research*, 45(5):768–778, 1997.

S. Martello and P. Toth. Knapsack Problems – Algorithms and Computer Implementations. *Wiley Interscience*, 1990.

G. Ottosson and E.S. Thorsteinsson. Linear Relaxation and Reduced-Cost Based Propagation of Continuous Variable Subscripts. *Proceedings of the CP-AI-OR'00*, pp. 129–138, 2000.

D. Pisinger. An expanding-core algorithm for the exact 0-1 knapsack problem. *European Journal of Operational Research*, 87:175–187, 1995.

D. Pisinger. An exact algorithm for large multiple knapsack problem. *European Journal of Operational Research*, 114:528–541, 1999.

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.

J.-C. Régin. A filtering algorithm for constraints of difference in CSPs, *Proc. of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pp. 362–367, 1994.

M. Sellmann and T. Fahle. Constraint Programming Based Lagrangian Relaxation for a Multimedia Application *Proceedings CP-AI-OR'01*, Ashford/UK, April 2001(a).

M. Sellmann and T. Fahle. Coupling Variable Fixing Algorithms for the Automatic Recording Problem. *9th Annual European Symposium on Algorithms (ESA 2001)*, Springer LNCS 2161: 134–145, 2001(b).

M. Trick. A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. *Proceedings of CP-AI-OR'01*, Ashford/UK, April, 2001.

P.R.C. Villela and C.T. Bornstein. An improved bound for the 0-1 knapsack problem. Report ES31-83, COPPE-Federal University of Rio de Janeiro, 1983.