Length-Lex Bounds Consistency for Knapsack Constraints[*]

Yuri Malitsky,[1] Meinolf Sellmann,[1] Willem-Jan van Hoeve[2]

[1] Brown University, Department of Computer Science
115 Waterman Street, P.O. Box 1910, Providence, RI 02912

[2] Carnegie Mellon University, Tepper School of Business
5000 Forbes Avenue, Pittsburgh, PA 15213

Abstract. Recently, a new domain store for set-variables has been proposed which totally orders all values in the domain of a set-variable based on cardinality and lexicography. Traditionally, knapsack constraints have been studied with respect to the required and possible set domain representation. For this domain-store efficient filtering algorithms achieving relaxed and approximated consistency are known. In this work, we study the complexity of achieving length-lex and approximated length-lex bounds consistency. We show that these strengthened levels of consistency can still be achieved in (pseudo-)polynomial time. In addition, we devise heuristic algorithms that work efficiently in practice.

1 Introduction

The constraint programming paradigm is inherently associated with the decomposition of a given problem into its constituting parts as given by the constraints describing it. Based on this decomposition, the standard solution scheme of constraint programming interleaves search and constraint propagation. The latter consists in constraint-specific filtering algorithms that remove inconsistent values from the variable domains. Thus, information from one constraint is propagated to the other constraints solely through variable domains.

While the decomposition allows the solver to apply filtering algorithms that are custom tailored for specific constraints, the main drawback of this scheme is that the information exchange via variable domains inherently weakens the ability to reason about the given problem. Consider for example the well-known example of three binary not-equal constraints working on three variables that all must be assigned either value A or value B. Clearly, no solution exists, but no constraint is able to convey information to the other constraints through the domain store that would make it possible to infer that the problem is infeasible.

Consequently, constraint programming research has looked for ways to strengthen inference by softening the effects of problem decomposition. The concept of global constraints has been an area of very productive research. In our example above, the All-Different constraint represents the conjunction of all three binary not-equal constraints. Other schemes such as CP-based Lagrangian relaxation [17] or CP-based Bender's decomposition [5, 9] have been proposed where constraints exchange dual or no-good

---

information on top of the traditional domain information. Recently, [1] proposed to use a multi-dimensional decision diagram as domain store which allows much more information to be represented and exchanged.

An alternative route is to consider more elaborate domains. For set variables, the traditional domain representation has been to specify a lower bound of *mandatory elements* and an upper bound of *possible elements*. In other words, the domain of a set-variable would be (partially) ordered based on the subset relation. This representation is equivalent to a representation through a number of binary variables.

As an easy way to strengthen the information captured, a variable representing the cardinality of the set could be added to this representation. In [13], Sadler and Gervet apply a *lexicographic* order to the domain of a set-variable. That is, the lower bound and upper bound become the lexicographically smallest and largest set that the variable can be assigned. In [8], Gervet and Van Hentenryck propose a *length-lexicographic* order for the domains of set variables. Domain elements (i.e., the sets that can potentially be assigned to the set-variable) are first ordered by increasing cardinality, while sets with equal cardinality are ordered lexicographically. The benefits of the length-lexicographic representation with respect to the traditional subset bounds representation are that it has less space requirements, unary constraints can be efficiently filtered, and it automatically breaks some symmetry.

In order to fully benefit from new domain representations such as the length-lexicographic set-variable domains, ways must be devised on how to exploit and strengthen the additional information efficiently. For the length-lexicographic representation, a first step in this direction was made by Dooms et al. [4], who presented domain filtering algorithms for *open constraints* where the set-variable representing the scope of an open constraint has a length-lexicographically ordered domain. For many important constraints involving set-variables, however, it remains an open question whether or not it is possible to efficiently filter the domains to some specified level of consistency when they are length-lexicographically ordered. Addressing this question with respect to knapsack constraints is the central topic of this paper.

Knapsack constraints have typically been defined over a number of binary variables which model whether an item is included in or excluded from the knapsack. An alternative way of modeling the constraint is by associating it with one set-variable. For the latter case, filtering algorithms for knapsack constraints achieving bounds-consistency for the subset-domain representation (which is equivalent to achieving generalized arc-consistency (GAC) for the binary variable representation) have been studied in [19, 14]. Since determining whether an item must be included or excluded in all feasible improving solutions is naturally NP-hard, the filtering algorithms either run in pseudo-polynomial time [14, 19], or they only achieve relaxed or approximated consistency [6, 11, 14, 16].

The length-lex domain representation provides information beyond the classical sets of required and possible items. A priori, it is not clear that the knapsack problem does not become strongly NP-hard when we require that the set of items must be assigned a value within given length-lex bounds. If this was the case, we would no longer be able to approximate the problem efficiently. In this paper, we show that there still exists a pseudo-polynomial filtering algorithm that establishes length-lex bounds-consistency for knapsacks. We also show how to transform this algorithm into a fully-polynomial approximation scheme and explain how this algorithm can be used to

obtain a polynomial-time algorithm which achieves approximated length-lex bounds-consistency for knapsack constraints.

While a complexity analysis of knapsack problems under length-lex constraints is interesting in its own right, in practice we often find that even polynomial-time filtering algorithms can be too heavy to pay-off within a search. For that reason, we also propose efficient heuristic filtering algorithms that communicate and exploit only the cardinality information embedded in the length-lex domain representation. We evaluate those heuristics empirically in the context of multi-knapsack problems where we find that exploiting cardinality information can effectively reduce the size of the search-tree.

2 Domain Representations for Set-Variables

We assume basic familiarity with constraint programming. Here we recall the basic definitions concerning constraint programming with set-variables [7].

A *set-variable* is a variable whose domain values are sets. We assume that the elements originate from a finite universe of elements. Because the number of possible values of a set-variable can be enormous (the size of a power set, in the worst case), one usually represents the domain of a set-variable S by a 'lower bound' $L(S)$ and an 'upper bound' $U(S)$ on the values that S can take.

A natural representation for the domain of a set-variable is based on the *subset ordering* of the universe. That is, the lower bound $L(S)$ represents all *mandatory* elements, while the upper bound $U(S)$ represents all *possible* elements, i.e., $D(S) = \{s \mid L(S) \subseteq s \subseteq U(S)\}$. We refer to this representation as the *subset* representation. In addition, at times also a lower bound $l(S)$ and upper bound $u(S)$ on the *cardinality* of S are maintained. We can add these two bounds to the subset representation for the domain of S, i.e., $D(S) = [L(S), U(S), l(S), u(S)] = \{s \mid L(S) \subseteq s \subseteq U(S), l(S) \leq |s| \leq u(S)\}$. We refer to this representation as the *subset-cardinality* representation.

An alternate representation is based on the *length-lexicographic* ordering of the universe [8] where the lower bound $L(S)$ represents the smallest set that can be assigned to S, while the upper bound $U(S)$ represents the largest set, i.e., $D(S) = \{s \mid L(S) \leq_{LL} s \leq_{LL} U(S)\}$. Here $\leq_{LL}$ denotes the length-lexicographic order. We refer to this representation as the *length-lex* representation.

Example 1. Let S be a set-variable representing a set of cardinality 2 or 3, in which element 4 is required, while any element from the set $\{1, 2, 3, 4, 5\}$ may appear. Using the subset-cardinality representation, $D(S) = [\{4\}, \{1, 2, 3, 4, 5\}, 2, 3]$. Note that the bounds of this representation do not correspond to feasible assignments for this variable.

Using the length-lex representation, we have $D(S) = [\{1, 4\}, \{3, 4, 5\}]$. Note that in this case, the bounds do correspond to feasible assignments for this variable. However, this representation also allows to assign sets that do not include element 4, for example, sets $\{1, 5\}$ and $\{2, 3\}$ are within the two specified bounds.

As pointed out in the example, a drawback of the length-lex representation is that it does not allow to represent (and exploit) mandatory elements directly. Therefore, we introduce an adapted representation that does allow to capture that information, combining the subset-cardinality and length-lex representations. We propose to maintain a set $R(S)$ of required elements, a set $P(S)$ of possible elements other than $R(S)$, and two sets $L_{lex}(S)$ and $U_{lex}(S)$ that denote the length-lexicographically smallest and largest

set that we can add to $R(S)$. In other words, the domain of a set-variable S is represented as $D(S) = [R(S), L_{lex}(S), U_{lex}(S), P(S)] = \{s \mid s = R(S) \cup t, L_{lex}(S) \leq_{LL} t \leq_{LL} U_{lex}(S), t \subseteq P(S)\}$. We finally define the shorthands $L(S) = R(S) \cup L_{lex}(S)$ and $U(S) = R(S) \cup U_{lex}(S)$. We refer to this representation as the length-lex* representation. [3]

Example 2. Continuing Example 1, the length-lex* representation gives $R(S) = \{4\}$, $L_{lex}(S) = \{1\}, U_{lex}(S) = \{3, 5\}$, and $P(S) = \{1, 2, 3, 5\}$. This defines the following domain (in length-lex order) for S: $\{1, 4\}$, $\{2, 4\}$, $\{3, 4\}$, $\{4, 5\}$, $\{1, 2, 4\}$, $\{1, 3, 4\}$, $\{1, 4, 5\}$, $\{2, 3, 4\}$, $\{2, 4, 5\}$, $\{3, 4, 5\}$.

For constraints involving set-variables, the filtering task is to increase the lower bounds and decrease the upper bounds of the domains. Ideally, we would like to achieve a specified level of consistency, for example bounds consistency or approximated bounds consistency. The respective definitions of these consistencies depend on the applied domain representation. For the length-lex* representation, we have:

Definition 1. *Let S denote a set-variable with length-lex* domain $D(S) = [R(S), L_{lex}(S), U_{lex}(S), P(S)]$. A constraint $C(S)$ is called* length-lex* bounds consistent *iff*

- $R(S) = \inf_{\subseteq}\{s \mid s \in D(S) \wedge s \in C(S)\}$,
- $P(S) = \sup_{\subseteq}\{s \mid s \in D(S) \wedge s \in C(S)\} \setminus R(S)$,
- $L_{lex}(S) = \min_{\leq_{LL}}\{s \mid s \in D(S) \wedge s \in C(S)\} \setminus R(S)$,
- $U_{lex}(S) = \max_{\leq_{LL}}\{s \mid s \in D(S) \wedge s \in C(S)\} \setminus R(S)$.

The notion of approximated length-lex* bounds consistency will be presented in Section 5.

3 The Knapsack Problem

In this section we fix notation that we use throughout the paper. First, we define the knapsack constraint $KP(S, p, B, w, C)$ representing the knapsack problem on a set of items denoted by a set-variable S defined on the universe of n items I, a profit vector p such that $p_i > 0$ is the profit of item $i \in I$, a weight vector w such that $w_i > 0$ is the weight of item $i \in I$, a lower bound B on the total profit, and a capacity C on the total weight of the knapsack. More formally, we have

$$KP(S, p, B, w, C) = \{s \mid s \in D(S), \sum_{i \in s} p_i \geq B, \sum_{i \in s} w_i \leq C\}.$$

As an alternative to the set-variable S, we can represent the set of items to include by a vector of binary variables x indexed by I, i.e., $(i \in S) \Leftrightarrow (x_i = 1)$. It is well-known that achieving generalized arc consistency with respect to the variables x corresponds to achieving subset bounds consistency with respect to the set-variable S [7].

---

[3] Note that our length-lex* representation is closely related to the 'hybrid' domain representation of Sadler and Gervet [13, Definition 3]. The difference is that the hybrid representation separates the lexicographic ordering and the bounds on the cardinality, while we treat them simultaneously using length-lex bounds. Furthermore, the hybrid representation allows the lexicographic bounds and the required elements to share elements, which we forbid.

4 Exact Pseudo-Polynomial Algorithms for Knapsack

Let us begin by ignoring the lexicographic bounds and assume that we are only given bounds on the number of items to include. The traditional way to achieve GAC for the binary variable representation of a knapsack constraint is by exploiting the dynamic programming (DP) principle. Since the maximum profit is achieved by either excluding or including any particular item, we have the following recursion equation: For all $k \in \mathbb{N}$ and $0 \leq q \leq \sum_i p_i$, the minimum weight $D_{k,q}$ needed to achieve profit q using only items in $\{1, \dots, k\}$ is

$$D_{k,q} \leftarrow \min\{D_{k-1,q}, D_{k-1,q-p_k} + w_k\}.$$

The maximum profit is then easily determined by finding the maximal q such that $D_{n,q} \leq C$. In [14], a filtering algorithm for knapsacks was developed that exploits Trick's well-known filtering technique for dynamic programs [19]. The main idea is to consider a dynamic program as a graph where each cell is a node and each node has exactly those predecessors as given by the dynamic programming recursion equation. For example, the predecessors of $D_{k,q}$ are $D_{k-1,q}$ and $D_{k-1,q-p_k}$. Edges are weighted by associating the edge from predecessor $D_{k-1,q}$ with weight 0 (as it does not cost anything to not include item k), and the edge from predecessor $D_{k-1,q-p_k}$ with weight w_k. This way, the values computed by the DP correspond directly to the shortest path distances from root-node $D_{0,0}$.

Moreover, every path from the root to some node $D_{n,q}$ corresponds directly to a knapsack solution and vice versa. We call such paths "admissible" if and only if $q \geq B$ and their length is lower or equal C. Now, by exploiting shorter path constraint filtering [15], we can shrink the graph by eliminating all edges in the graph which can not be visited by any admissible path. To this end, we introduce an artificial sink-node t that has predecessors $D_{n,q}$ for all $q \geq B$. Then, shortest path distances from the root to all nodes and the corresponding shortest-path distances from all nodes to the sink-node t can be used to determine which edges can still be used on some admissible path [15]. Finally, we infer that item k must (cannot) be included in any feasible and improving knapsack iff for all q the only predecessor of $D_{k,q}$ in the shrunken graph is $D_{k-1,q-p_k}$ ($D_{k-1,q}$) [19].

In the presence of constraints limiting the cardinality to fall into a given interval $[l, u]$, the following analogous dynamic programming recursion solves the knapsack problem with cardinality bounds:

$$W_{k,q,c} \leftarrow \min\{W_{k-1,q,c}, W_{k-1,q-p_k,c-1} + w_k\}.$$

Again, since item k is either included or excluded in the optimal solution, $W_{k,q,c}$ gives the minimum weight needed to achieve a profit of exactly $q \in \mathbb{N}$ when using exactly c of the first k items. And again, based on this recursion the optimum is easily determined by finding the maximum q and $c \in [l, u]$ such that $W_{n,q,c} \leq C$. By introducing an artificial sink t with predecessors $W_{n,q,c}$ for all $q \geq B$ and $c \in [l, u]$, we exploit once more shorter path filtering to determine all items that must or cannot be included by a knapsack constraint augmented by a constraint on the cardinality. Furthermore, we can infer new bounds on the cardinalities by finding the minimum and maximum values for c for which there exists a predecessor $W_{n,q,c}$ of t in the shrunken

graph. The total runtime of this algorithm is in $O(n^2 u ||p||_\infty) = O(n^3 ||p||_\infty)$, where $||p||_\infty = \max_i p_i$.

Now, when we want to achieve length-lex* bounds consistency for knapsack constraints in the set-variable representation, we can exploit the above algorithm by decomposing the constraint as follows. Let S be a set variable with length-lex* domain $D(S) = [R(S), L_{lex}(S), U_{lex}(S), P(S)]$, based on a universe of items $\{1, \ldots, n\}$ (recall we use shorthands $L(S) = R(S) \cup L_{lex}(S)$ and $U(S) = R(S) \cup U_{lex}(S)$). Then

$$KP(S, p, B, w, C) \Leftrightarrow KP(S^1, p, B, w, C) \vee KP(S^2, p, B, w, C) \vee KP(S^3, p, B, w, C), \tag{1}$$

where S^1, S^2, and S^3 are set-variables with respective length-lex* domains

$$D(S^1) = D(S) \cap [L(S), \min_{\leq_{LL}}(\{n - |L(S)| + 1, \ldots, n\}, U(S))],$$
$$D(S^2) = D(S) \cap [\{1, \ldots, |L(S)| + 1\}, \{n - |U(S)| + 2, \ldots, n\}],$$
$$D(S^3) = D(S) \cap [\max_{\leq_{LL}}(L(S), \{1, \ldots, |U(S)|\}), U(S)].$$

Note that S^1 and S^3 have real lexicographic bounds but are fixed in cardinality with $|S^1| = |L(S)|$ and $|S^3| = |U(S)|$. On the other hand, S^2 has only trivial lexicographic bounds, and it holds that $|L(S)| < |S^2| < |U(S)|$ (provided that $|U(S)| - |L(S)| \geq 2$). Therefore, for S^2 we can exploit the algorithm that we sketched above. Thus, for a complete pseudo-polynomial length-lex bounds consistency algorithm, we only lack a pseudo-polynomial filtering algorithm for knapsacks with fixed cardinality and arbitrary lexicographic bounds.

So let us consider the following problem: Given a natural number n, a profit vector $p \in \mathbb{Q}^n$, a profit threshold $B \in \mathbb{N}$, a weight vector $w \in \mathbb{Q}^n$, a capacity $C \in \mathbb{N}$, a fixed cardinality $\kappa \in \mathbb{N}$, and lexicographic bounds $L, U \subseteq \{1, \ldots, n\}$, find a solution $x \in \{0, 1\}^n$ such that

$$p^T x \geq B \qquad\qquad\qquad w^T x \leq C \tag{2}$$
$$1^T x = \kappa \qquad\qquad L \leq_{lex} \{i \mid x_i = 1\} \leq_{lex} U \tag{3}$$
$$x \in \{0, 1\}^n. \tag{4}$$

By using a three-dimensional DP like before, we can directly enforce the capacity and profit restrictions (simply by only allowing nodes $W_{n,q,c}$ to connect to the sink-node t for which $c = \kappa$ and $q \geq B$). The filtering problem can then be addressed by identifying edges in the DP-induced graph for which there exists no admissible path from the root to the sink, whereby admissibility now enforces both a path-length lower or equal C *and* that the corresponding knapsack solution is a set of items S for which $L \leq_{lex} S \leq_{lex} U$. To this end, we intend to reuse the idea of shorter-path constraint filtering. However, a simple forward and backward shortest path computation is no longer sufficient because the concatenation of a path from the source to a node in the graph and a path from that node to the sink may violate the lexicographic bounds.

Note that both L and U define (potentially non-admissible) paths in the DP-induced graph that we denote with π_L and π_U, respectively. Conversely, for any path π from the root to any node in the DP, we can define a corresponding set S of items that the path includes in the knapsack: $S_\pi \leftarrow \{k \mid (W_{k-1,q-p_k,c-1}, W_{k,q,c}) \in \pi\}$.

In order to identify exactly those nodes in the graph that have no admissible paths running through them, it will be important to know the shortest path distance from the root to a given node when the choices implied by that path π already ensure that the resulting set S_π must obey the lexicographic bounds L, U. Formally:

Definition 2. *For a path π from the root to $W_{k,q,c}$, we write $L <_{lex} S_\pi$ (or $S_\pi <_{lex} U$) if and only if for all $S \subseteq \{1, \ldots, n\}$ such that $S \cap (S_\pi \cup \{k + 1, \ldots, n\}) = S$ and $|S| = \kappa$ it holds that $L <_{lex} S$ ($S <_{lex} U$).*

Conversely, we will also need to argue about paths from nodes in the DP-induced graph to the sink-node t:

Definition 3. *For a path π from $W_{k,q,c}$ to t, we write $L \leq_{lex} S_\pi$ (or $S_\pi \leq_{lex} U$) if and only if for $T \leftarrow S_\pi \cup (L \cap \{1, \ldots, k\})$ ($T \leftarrow S_\pi \cup (U \cap \{1, \ldots, k\})$) it holds that $|T| = \kappa$ and $L \leq_{lex} T$ ($T \leq_{lex} U$).*

To make our task easier, we may assume that the first item is a member of L (otherwise the item is disallowed and can be removed from consideration), and that the first item is not in U (as otherwise the item must be taken and could also be removed from the problem). Then, for all nodes but the root, we distinguish three situations.

Remark 1.   1. For all nodes $W_{k,q,c}$ that are neither on π_L nor on π_U, a shortest admissible path from the root to t that visits this node obviously decomposes into a part from the root to the given node π_1, and from the node to the sink π_2. Since the current node is neither on π_L nor on π_U, we know that $L <_{lex} S_{\pi_1} <_{lex} U$.
   2. For a node $W_{k,q,c}$ on π_L, on top of option 1, the shortest admissible path from root to t through this node may follow π_L from the root to the node, and some path π_2 from the node to t with $L \leq_{lex} S_{\pi_2}$.
   3. For a node $W_{k,q,c}$ on π_U, on top of option 1, the shortest admissible path from root to t through this node may follow π_U from the root to the node, and some path π_2 from the node to t with $S_{\pi_2} \leq_{lex} U$.
   Note that options 2 and 3 may occur at the same time.

Consequently, we can compute the length of the shortest admissible path through a given node $W_{k,q,c}$, if we know the following six quantities:

- For $W_{k,q,c} \in \pi_L$, $M^1_{k,q,c}$ gives the distance from the root to $W_{k,q,c}$ when following π_L, that is $M^1_{k,q,c} \leftarrow \sum_{i \in L, i \leq k} w_i$. For $W_{k,q,c} \notin \pi_L$, we set $M^1_{k,q,c} \leftarrow \infty$.
- For $W_{k,q,c} \in \pi_U$, $M^2_{k,q,c}$ gives the distance from the root to $W_{k,q,c}$ when following π_U, that is $M^2_{k,q,c} \leftarrow \sum_{i \in U, i \leq k} w_i$. For $W_{k,q,c} \notin \pi_U$, we set $M^2_{k,q,c} \leftarrow \infty$.
- For arbitrary nodes $W_{k,q,c}$, $M^3_{k,q,c}$ gives the length of the shortest path π from the root to $W_{k,q,c}$ with $L <_{lex} S_\pi <_{lex} U$.
- For arbitrary nodes $W_{k,q,c}$, $M^4_{k,q,c}$ gives the length of the shortest path π from $W_{k,q,c}$ to t.
- For arbitrary nodes $W_{k,q,c}$, $M^5_{k,q,c}$ gives the length of the shortest path π from $W_{k,q,c}$ to t with $L \leq_{lex} S_\pi$.
- For arbitrary nodes $W_{k,q,c}$, $M^6_{k,q,c}$ gives the length of the shortest path π from $W_{k,q,c}$ to t with $S_\pi \leq_{lex} U$.

Lemma 1. – *The length of a shortest admissible path through an edge $(W_{k,q,c}, W_{k+1,q,c})$ is*

$$\min\{M^3_{k,q,c} + M^4_{k+1,q,c}, M^1_{k,q,c} + D^1_{k,q,c}, M^2_{k,q,c} + D^2_{k,q,c}\},$$

where $D^1_{k,q,c} = M^4_{k+1,q,c}$ if $k+1 \in L$ and $D^1_{k,q,c} = M^5_{k+1,q,c}$ otherwise, and $D^2_{k,q,c} = M^6_{k+1,q,c}$ if $k+1 \notin U$ and $D^2_{k,q,c} = \infty$ otherwise.

– *The length of a shortest admissible path through an edge $(W_{k,q,c}, W_{k+1,q+p_{k+1},c+1})$ is*

$$\min\{M^3_{k,q,c} + M^4_{k+1,q+p_{k+1},c+1} + w_{k+1}, M^1_{k,q,c} + E^1_{k,q,c} + w_{k+1}, M^2_{k,q,c} + E^2_{k,q,c} + w_{k+1}\},$$

where $E^1_{k,q,c} = M^5_{k+1,q+p_{k+1},c+1}$ if $k+1 \in L$ and $E^1_{k,q,c} = \infty$ otherwise, and $E^2_{k,q,c} = M^4_{k+1,q+p_{k+1},c+1}$ if $k+1 \notin U$ and $E^2_{k,q,c} = M^6_{k+1,q+p_{k+1},c+1}$ otherwise.

Proof. Assume $W_{k,q,c} \in \pi_L$. Denote with π_1 the path from the root to $W_{k,q,c}$ by following π_L. Since $1 \in L$ and $1 \notin U$, we know that $S_{\pi_1} <_{lex} U$. Consequently, it is sufficient for quantity M^5 to consider the lower lexicographic bound only when combined with M^1. The analogue holds for the combination of M^2 and M^6. With this observation, the lemma follows from Remark 1. $\square$

Consequently, our task is to compute quantities $M^1, \dots, M^6$. For M^1 and M^2, this is straightforward. For $M^3, \dots, M^6$, in the following we devise recursion equations which allow us to compute them by means of dynamic programming.

Recall that M^3 measures the shortest path distance from the root when this path already ensures that the final path will strictly obey both lexicographic bounds (let us call such a path an M^3-path). When considering the inclusion or exclusion of item k, we can either use an M^3-path to reach the predecessor of a node, in which case we know that any continuation results in an M^3-path. Alternatively, we can consider a predecessor node on π_L when the exclusion of k results in an M^3 path. Analogously, we can consider a predecessor node on π_U when the inclusion of k results in an M^3 path. Consequently, we have the following recursion equation:

$$M^3_{k,q,c} = \min\{M^3_{k-1,q,c}, M^3_{k-1,q-p_k,c-1} + w_k, A^1_{k,q,c}, A^2_{k,q,c}\},$$

whereby $A^1_{k,q,c} = M^1_{k-1,q,c}$ if $k \in L$ and $A^1_{k,q,c} = \infty$ otherwise, and $A^2_{k,q,c} = M^2_{k-1,q-p_k,c-1} + w_k$ if $k \notin U$ and $A^2_{k,q,c} = \infty$ otherwise.

Quantity M^4 plainly computes the shortest path distance to the sink t, so the common recursion equation works without modification:

$$M^4_{k,q,c} = \min\{M^4_{k+1,q,c}, M^4_{k+1,q+p_{k+1},c+1} + w_{k+1}\}.$$

Quantity M^5 measures the distance to the sink-node t when only paths are allowed that obey the lexicographic lower bound when we prepend the lower-bound path to the current node. When considering the exclusion of an item $k+1$ with $k+1 \in L$, we are sure to strictly obey the lexicographic lower bound and can therefore use the unrestricted shortest path distance to the sink of the corresponding successor node. Consequently, we have the following recursion equation:

$$M^5_{k,q,c} = \min\{B_{k,q,c}, M^5_{k+1,q,c}\},$$

whereby $B_{k,q,c} = \min\{M^4_{k+1,q,c}, M^5_{k+1,q+p_{k+1},c+1} + w_{k+1}\}$ if $k+1 \in L$ and $B_{k,q,c} = \infty$ otherwise. The analogue argument for M^6 gives:

$$M^6_{k,q,c} = \min\{C_{k,q,c}, M^6_{k+1,q+p_{k+1},c+1} + w_{k+1}\},$$

whereby $C_{k,q,c} = \min\{M^4_{k+1,q+p_{k+1},c+1} + w_{k+1}, M^6_{k+1,q,c}\}$ if $k+1 \notin U$ and $C_{k,q,c} = \infty$ otherwise.

With these results, we are now able to prove the following theorem:

Theorem 1. *Let S be a set-variable with length-lex* domain based on a universe of elements $\{1, \ldots, n\}$. For a Knapsack constraint $KP(S, p, B, w, C)$, length-lex* bounds consistency can be achieved in time $O(n^3 ||p||_\infty)$.*

Proof. We first decompose the constraint according to Equation 1. As discussed earlier, we can identify all possible and required items for $KP(S^2, p, B, w, C)$ in pseudo-polynomial time. Next we consider $KP(S^1, p, B, w, C)$ and $KP(S^3, p, B, w, C)$ and filter edges according to the algorithm sketched above (whereby the lexicographic upper or lower bound are set to the trivial bound for the given cardinality $\kappa \leftarrow |L(S)|$ or $\kappa \leftarrow |U(S)|$ when $|L(S)| < |U(S)|$). We set up the DP-induced graph and compute quantities $M^1, \ldots, M^6$ for all nodes. Then, we filter all nodes and edges from the graph which cannot be visited by any admissible path. Using Trick's DP-filtering technique, this allows us to identify all items which must or cannot be part of any feasible improving solution for $KP(S^1, p, B, w, C)$ and $KP(S^3, p, B, w, C)$.

In this way we also determine whether there exist admissible paths at the cardinality bounds at all, i.e., whether the constraints can still be satisfied or not. If this is the case, in order to compute a new lexicographic lower bound at the lower cardinality bound, we simply include the first item if that is still possible after filtering edges from the graph. Then we filter again and try to include the next item and so forth. The correctness of the edge-filtering algorithm guarantees that we compute an admissible path π such that S_π is the lexicographically smallest feasible and improving solution with $|S_\pi| = |L(S)|$. For the new lexicographic upper bound we proceed analogously.

If we find that one of the two constraints is not satisfiable anymore, then we use $KP(S^2, p, B, w, C)$ again to determine a new lower and/or upper bound on the cardinality. If one of the cardinality bounds are updated, we repeat the computation of a new lexicographical lower and/or upper bound as before.

The total runtime of this algorithm is dominated by the computation of new lexicographical upper and lower bounds which require up to $|U(S)|$ calls to the edge-filtering algorithm whose runtime is determined by the size of the DP-induced graph which is in $O(n|U(S)|||p||_\infty)$. The total runtime is therefore in $O(n|U(S)|^2||p||_\infty) = O(n^3||p||_\infty)$. $\qquad\square$

5   Approximated Length-Lex Bounds Consistency for Knapsack constraints

The results of the previous section show that the fully polynomial-time approximeability of knapsack problems is not affected by additional length-lex bounds constraints. We can utilize our approximation scheme to achieve approximated length-lex* bounds consistency for knapsack constraints in the spirit of [14]:

Definition 4. *Let S denote a set-variable with length-lex* domain $D(S) = [R(S), L_{lex}(S), U_{lex}(S), P(S)]$. The knapsack constraint $\mathrm{KP}(S, p, B, w, C)$ is called ε-length-lex* bounds consistent *when it holds:*

- $P^*[i \in S] \geq B - \varepsilon P^*$, *for all* $i \in P(S)$,
- $P^*[i \notin S] < B - \varepsilon P^*$, *for all* $i \in R(S)$,
- $P^*[S = R(S) \cup L_{lex}(S)] \geq B - \varepsilon P^*$ *and* $P^*[S = R(S) \cup U_{lex}(S)] \geq B - \varepsilon P^*$,

where P^* gives the optimal knapsack solution (potentially under the additional constraints given in brackets).

Theorem 2. *Approximated ε-length-lex* bounds consistency for knapsack constraints can be achieved in time $O(\frac{n^4}{\varepsilon})$.*

Proof. In this proof we again use the shorthands $L(S) = R(S) \cup L_{lex}(S)$ and $U(S) = R(S) \cup U_{lex}(S)$.

We apply the standard approach from [10] for transforming a dynamic program into a fully polynomial-time approximation scheme (FPTAS): We scale the profits by setting $\tilde{p}_i \leftarrow \lfloor \frac{p_i}{K} \rfloor$ for $K \leftarrow \frac{\varepsilon ||p||_\infty}{|U(S)|}$. Then, we invoke our pseudo-polynomial filtering algorithm on $KP(S, \tilde{p}, B - \varepsilon ||p||_\infty, w, C)$. Note that $||\tilde{p}||_\infty \leq \frac{|U(S)|}{\varepsilon}$. Therefore, the algorithm runs in time $O(n^2 |U(S)| ||\tilde{p}||_\infty) = O(\frac{n^2 |U(S)|^2}{\varepsilon})$.

We show that the algorithm is sound and achieves ε-length-lex* bounds consistency.
Soundness: Assume our algorithm excludes an item s from $P(S)$. It does so only when there exists no admissible path that includes the item, which is the same as to say that there exists no admissible solution to $KP(S, \tilde{p}, \frac{B - \varepsilon ||p||_\infty}{K}, w, C)$ that includes the item. Denote with $\tilde{S} \in D(S)$ the solution with $s \in \tilde{S}$ that maximizes $\tilde{P}^*[s \in S] = \sum_{i \in \tilde{S}} \tilde{p}_i$ while $\sum_{i \in \tilde{S}} w_i \leq C$. Furthermore, denote with $S^* \in D(S)$ the solution with $s \in S^*$ that maximizes $P^*[s \in S] = \sum_{i \in S^*} p_i$ while $\sum_{i \in S^*} w_i \leq C$. It holds:

$$B - \varepsilon ||p||_\infty > K \tilde{P}^*[i \in S] \tag{5}$$

$$\geq \sum_{i \in \tilde{S}} p_i - K|U(S)| \geq \sum_{i \in \tilde{S}} p_i - \varepsilon ||p||_\infty \tag{6}$$

Therefore, $P^*[s \in S] = \sum_{i \in \tilde{S}} p_i < B$, which means it is sound to remove item s from consideration. The analogous argument holds for items that are included by our algorithm. Next, our algorithm computes length-lex lower and upper bounds $L_{lex}(S), U_{Lex}(S)$ on the undecided items such that no set lower than $L(S) = R(S) \cup L_{lex}(S)$ and no set larger than $U(S) = R(S) \cup U_{lex}(S)$ is admissible for $KP(S, \tilde{p}, \frac{B - \varepsilon ||p||_\infty}{K}, w, C)$. The same argument as before shows that no set lower than $L(S)$ or larger than $U(S)$ can then be admissible for $KP(S, p, B, w, C)$.
Completeness: Assume for some item s it holds that $P^*[s \in S] < B - \varepsilon P^*$. Therefore, for all $\tilde{S} \in D(S)$ with $s \in \tilde{S}$ and $\sum_{i \in \tilde{S}} w_i \leq C$ it holds that $\sum_{i \in \tilde{S}} p_i < B - \varepsilon P^* \leq B - \varepsilon ||p||_\infty$. Thus:

$$B - \varepsilon ||p||_\infty > \sum_{i \in \tilde{S}} \tilde{p}_i \geq K \sum_{i \in \tilde{S}} \tilde{p}_i$$

for all $\tilde{S}$, and therefore s is removed from $P(S)$. The analogous results follows for items that must be included. For the length-lex bounds, finally, it holds that they define admissible solutions for $KP(S, \tilde{p}, \frac{B-\varepsilon||p||_\infty}{K}, w, C)$. It holds:

$$B - \varepsilon||p||_\infty \leq K \sum_{i \in L(S)} \tilde{p}_i \leq \sum_{i \in L(S)} p_i = P^*[S = L(S)].$$

And the analogue is true for the set $U(S)$. $\square$

6 Fast Heuristic Filtering Algorithms for Knapsacks with Bounded Cardinalities

We have seen that cardinality and lexicographic information can be inferred and taken into account for knapsack constraints without compromising the fully polynomial-time approximeability of the problem. However, although polynomial, a runtime in $O(n^4)$ is not practically appealing in light of the delicate trade-off between the time to perform this type of inference and the value of the additional information gained by it. In order to make inference faster, we may decide that we only want to reason about the cardinality of the final set of items included in the knapsack. We presented an exact pseudo-polynomial time algorithm for this task in Section 4. In an effort to reduce the filtering-time, in this section we devise a heuristic algorithm which runs in linear time.

6.1 Lagrangian Relaxation-based Cardinality Bounds

To derive cardinality bounds, as we did earlier in [18], we consider the Lagrangian relaxation of the knapsack problem. In linear programming, it is well-known that the optimal dual value for the capacity constraint is the efficiency (the profit over weight) of the critical item s, which is defined as the first item in the efficiency ordering whose inclusion overloads the knapsack: $s \leftarrow \min\{s' \mid \sum_{i=1}^{s'} w_i > C\}$, whereby $i < j$ implies $p_i/w_i \geq p_j/w_j$. Using this value as Lagrangian multiplier, we are left with the following relaxed problem: maximize $p^T x - (w^T x - C)p_s/w_s = (p - w p_s/w_s)^T x + C p_s/w_s$ such that $x_i \in \{0, 1\}$. Obviously, the maximum is obtained by setting $x_i \leftarrow 1$ if $\tilde{p}_i \leftarrow p_i - w_i p_s/w_s > 0$, and $x_i \leftarrow 0$ if $\tilde{p}_i < 0$ (for $\tilde{p}_i = 0$ we can set x_i arbitrarily). By this setting, we obtain a valid upper bound U on the profit that can be achieved. If $U < B$, we can backtrack right away as the current subproblem cannot have any improving feasible solutions. Otherwise, we would like to infer which items must be included/excluded as they must/cannot be part of any improving feasible solution. Moreover, we would like to tighten the bounds on the number of items that must/can be included in the knapsack.

When sorting items according to decreasing Lagrangian profits $\tilde{p}_i$, we can easily deduce lower and upper bounds on the number of items that must/can be included:

$$l \leftarrow \max\{l, \min\{l' \mid \sum_{i=1}^{l'} \tilde{p}_i \geq B - C\frac{p_s}{w_s}\}\},$$

whereby we assume that $i < j$ implies $\tilde{p}_i \geq \tilde{p}_j$. Analogously, we set

$$u \leftarrow \min\{u, \max\{u' \mid \sum_{i=1}^{u'} \tilde{p}_i \geq B - C\frac{p_s}{w_s}\}\}.$$

Input: set S with associated profit vector $\tilde{p}$ and a bound $\tilde{B}$.
Pick a random element r in S.
Set $L \leftarrow \{i \in S \mid \tilde{p}_i \geq \tilde{p}_r\}$ and $R \leftarrow \{i \in S \mid \tilde{p}_i < \tilde{p}_r\}$.
$\tilde{p}(L) \leftarrow \sum_{i \in L} \tilde{p}_i$
if $\tilde{p}(L) \geq \tilde{B}$ **then**
     | **return** lowerBound$(L, \tilde{p}, \tilde{B})$
else
     | **return** $|L|$+lowerBound$(R, \tilde{p}, \tilde{B} - \tilde{p}(L))$

Algorithm 1: Linear-time algorithm to determine a lower bound on the cardinality.

The effort for the above update is obviously dominated by sorting the items, which can be done in time $O(n \log n)$. However, a complete sorting is actually not necessary. Just like the critical item s of a knapsack instance can be computed in linear time [3], so can the new cardinality bounds l and u. In Algorithm 1, we show how a lower bound on the cardinality can be computed in expected linear time. The algorithm works like a quick-sort algorithm, whereby only one part of the items needs to be investigated recursively. According to the master theorem for recursive algorithms [2, Section 4.3, 4.4], this lowers the time from $O(n \log n)$ to $O(n)$ when we assume that, on average, the set of items is cut in half in each recursion iteration. While this works well in practice, in theory we can even guarantee a linear runtime by replacing the random choice of the splitting item by the median item, where the median can be computed in linear time [2].

Of course, the Lagrangian relaxation also allows us to filter items by exploiting the idea of CP-based Lagrangian relaxation [17]. In our case, filtering is particularly easy as items that must be included have $\tilde{p}_i > B - U$. Those that cannot be included have $\tilde{p}_i < U - B$. In case that the external cardinality bounds are tight (this happens when $\tilde{p}_l < 0$ or $\tilde{p}_u > 0$), we can even decide that an item must be included when $\tilde{p}_i - \tilde{p}_{l+1} > B - U$ when the lower cardinality bound is tight, and that an item must be excluded when $\tilde{p}_i - \tilde{p}_u < B - U$ when the upper cardinality bound is tight.

6.2 Redundant Knapsack constraints

In terms of running time, the above linear time algorithm that heuristically filters knapsack constraints and exploits and provides upper and lower bounds on the cardinality is already much more appealing than the exact or approximate algorithms devised earlier. However, we can do even more: In [11], an algorithm for the propagation of knapsack constraints was devised which runs in amortized expected sublinear time. The question arises how this algorithm can be exploited to reason about knapsack cardinalities at the same time as it infers which items must or cannot be included in any feasible improving solution. A simple option is to post redundant knapsack constraints: If we are given the conjunction $KP(S, p, B, w, C) \wedge (l \leq |S| \leq u)$, we can post the following three traditional knapsack constraints (whereby $x_1, \ldots, x_n$ are binary variables): $KP(x_1, \ldots, x_n, p, B, w, C)$, $KP(x_1, \ldots, x_n, (1, \ldots, 1)^T, l, w, C)$, and $KP(x_1, \ldots, x_n, p, B, (1, \ldots, 1)^T, u)$. Note that the different constraints do not only allow us to perform filtering in the item variables, they also allow us to infer strengthened bound on the cardinalities. For example, the floor of the linear upper bound computed for the propagation of $KP(x_1, \ldots, x_n, (1, \ldots, 1)^T, l, w, C)$ gives a valid upper bound on the number of items that can be included in any feasible and improving solution.

Class	#Items	Lagrangian		KP		KP+Card	
		Time	**#CPs**	**Time**	**#CPs**	**Time**	**#CPs**
P1	15	1.89	26.7k	1.51	24.1k	2.19	22.4k
	20	2.40k	31.4M	888.51	12.6M	978.79	8.62M
P2	15	5.98	85.4k	1.51	22.1k	2.34	21.6k
	20	4.26k	57.8M	414.25	5.48M	468.65	3.74M
P3	15	2.41	32.2k	0.33	4.88k	0.54	4.84k
	20	264.33	3.32M	12.85	0.19M	22.37	0.19M

Table 1. *Average running time (seconds) and the average number of choice points over 20 instances with five knapsack constraints and 15 or 20 items using three different models.*

7 Experimental Results

Despite the fact that the algorithms developed in Sections 4 and 5 are polynomial, their comparably large computation costs render them impractical within backtrack search where we face a delicate trade-off between inference efficiency and effectiveness. To assess whether communicating information beyond the traditional inclusion or exclusion of items, we therefore implemented the heuristic algorithms for reasoning about knapsacks with cardinality constraints.

As our benchmark, we use multi-knapsack problems where we have to distribute a set of items over multiple knapsacks while the capacity restrictions on the individual knapsacks must be respected. We aim at maximizing the profit of the knapsack that is assigned the least profit. Problems are generated using the code from Pisinger [12]. We distinguish three different problem classes:

P1: Multi-Knapsack problems where all knapsacks use the same profit and weight vector. Constraints differ in the available capacity for each knapsack.
P2: Multi-Knapsack problems where all knapsacks use the same profit vector. Constraints differ in the weight vector and the available capacity for each knapsack.
P3: Multi-Knapsack problems where all knapsacks use different profit and weight vectors.

In each algorithm, we branch on the item that has the least knapsacks left to be assigned to, and we assign it to that knapsack that has been assigned the least profit yet. We compare three different models:

Class	#Items	KP vs KP+Card				Lagrangian vs KP+Card			
		Time		**#CPs**		**Time**		**#CPs**	
		avg	**var**	**avg**	**var**	**avg**	**var**	**avg**	**var**
P1	15	-43.4	0.8	7.0	0.3	-4.6	16.4	24.9	10.7
	20	-32.4	9.6	18.1	3.0	51.0	12.1	67.2	6.1
P2	15	-54.2	0.6	1.2	0.0	64.1	5.4	77.5	3.1
	20	-50.6	4.5	8.9	1.5	92.3	0.6	95.8	0.2
P3	15	-55.3	1.1	1.2	0.2	81.5	5.2	90.2	2.1
	20	-67.3	0.3	0.6	0.0	-353	3.9k	-284	2.8k

Table 2. *Average (avg) percent difference in running times and choice points as well as their variance (var) when comparing models on a collection of multi-knapsack problems. A positive value states the strategy listed first is the given percentage larger.*

- **KP**: the plain knapsack model where each knapsack constraint is propagated by the expected sublinear-time algorithm introduced in [11]. The partitioning of items is enforced by introducing item variables, whereby each of those variables has the indices of the knapsacks plus a dummy index for left-over items as its domain.
- **KP + Card:** the model where redundant knapsack constraints exploit cardinality information. Each knapsack is modelled by three constraints, one for the actual knapsack, one for the combination of cardinality upper bound and the knapsack's profit, and one for the combination of the original weights in combination with the cardinality lower bound. All knapsacks are propagated by the algorithm from [11]. The partitioning of items is enforced by a global cardinality constraint which exploits and strengthens the cardinality bounds on the knapsacks.
- **Lagrangian:** the model where we use a Lagrangian bound to propagate knapsack constraints and infer knapsack cardinalities. The partitioning of items is again enforced by a global cardinality constraint.

All experiments were run on an AMD Athlon 64 X2 Dual Core Processor 3800+ using Ilog Solver 6.5. In Tables 1 and 2 we show the average runtime and choice points on collections of 20 instances in the different banchmark classes P1, P2, and P3. We see how exploiting cardinality information effectively reduces the number of choice points. This is generally highly desirable as a more effective inference mechanism leaves less room for mistakes when organizing the search. For multi-knapsack problems, the trade-off between inference time and effectiveness is not in favor of even slightly more costly inference, and we observe that the plain KP model works fastest in all cases. Note that, in this model, inference works in expected sublinear time, while in the two other models global cardinality constraints need to be propagated to infer new cardinality bounds on the knapsacks. The Lagrangian model also suffers from a linear-time filtering routine for knapsack, and we see that it cannot compete with with KP and KP+Card.

We were curious to see whether the reductions in choice points become more important as problem instances become even harder. In Table 3 we compare KP and KP+Card on ten 22 item knapsack problems. We observe that the exchange of cardinality information is becoming more and more competitive, and for even harder problem instances we expect that the more costly yet more effective inference will eventually become beneficial. The results show that, with increasing difficulty of the problem, the cardinality constraints significantly boost the performance of the algorithm, providing an average decrease of more than 30% in the number of choice points. For more general problems, where knapsack constraints are mixed with other constraints, this reduction may be very beneficial.

ID	KP		KP+Card	
	Time	**#CPs**	**Time**	**#CPs**
1	4.5K	52.2M	5.4K	39.5M
2	1.4K	17.2M	2.6K	16.7M
3	2.4K	31.7M	2.4K	19.6M
4	1.8K	20.8M	2.9K	20.7M
5	15.3K	196.2M	17.6K	136.3M
6	0.5K	6.7M	0.5K	4.0M
7	4.3K	55.8M	3.1K	23.6M
8	2.4K	33.2M	2.6K	22.1M
9	3.2K	41.9M	4.5K	35.9M
10	4.3K	56.5M	3.1K	24.9M
avg	4.0K	51.2M	4.5K	34.3M

Table 3. *Running times (seconds) and the number of choice points for 10 instances with five knapsack constraints and 22 items using the KP and KP+Card models.*

8   Conclusions

We studied the complexity of knapsack constraints with length-lex domains and showed that the problem remains fully polynomial-time approximeable. Based on this result, we showed how ε-approximate length-lex bounds consistency for knapsacks can be achieved in time $O(n^4/\varepsilon)$. Compromising inference effectiveness for efficiency, we provided heuristic filtering algorithms for knapsack constraints that incorporate cardinality information only. Experiments on multi-knapsack problems showed that these algorithms effectively reduce the number of choice points. Whether or not this reduction is worthwhile will depend on the concrete problem that needs to be solved.

References

1. H.R. Andersen, T. Hadzic, J.N. Hooker, and P. Tiedemann. A Constraint Store Based on Multivalued Decision Diagrams. In *Proceedings of CP*, volume 4741 of *LNCS*, pages 118–132. Springer, 2007.
2. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms (Second Edition)*. MIT Press / McGraw-Hill, 2001.
3. G. Dantzig. Discrete variable extremum problems. *Operations Research*, 5:226–277, 1957.
4. G. Dooms, L. Mercier, P. Van Hentenryck, W.-J. van Hoeve, and L. Michel. Length-Lex Open Constraints. Technical Report CS-07-09, Brown University, 2007.
5. A. Eremin and M. Wallace. Hybrid Benders Decomposition Algorithms in Constraint Logic Programming. In *Proceedings of CP*, volume 2239 of *LNCS*, pages 1–15. Springer, 2001.
6. T. Fahle and M. Sellmann. Cost Based Filtering for the Constrained Knapsack Problem. *Annals of Operations Research*, 115(1):73–93, 2002.
7. C. Gervet. Constraints over structured domains. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 17. Elsevier, 2006.
8. C. Gervet and P. Van Hentenryck. Length-lex ordering for set CSPs. In *Proceedings of AAAI*, 2006.
9. J.N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(33-60):22, 2003.
10. O.H. Ibarra and C.E. Kim. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. *Journal of the ACM*, 22(4):463–468, 1975.
11. I. Katriel, M. Sellmann, E. Upfal, and P. Van Hentenryck. Propagating Knapsack Constraints in Sublinear Time. In *Proceedings of AAAI*. AAAI Press, 2007.
12. D. Pisinger. Where are the hard knapsack problems? *Computers and Operations Research*, 32:2271–2282, 2005.
13. A. Sadler and C. Gervet. Enhancing set constraint solvers with lexicographic bounds. *Journal of Heuristics*, 14(1):23–67, 2008.
14. M. Sellmann. Approximated Consistency for Knapsack Constraints. In *Proceedings of CP*, volume 2833 of *LNCS*, pages 679–693. Springer, 2003.
15. M. Sellmann. Cost-Based Filtering for Shorter Path Constraints. In *Proceedings of CP*, volume 2833 of *LNCS*, pages 694–708. Springer, 2003.
16. M. Sellmann. The Practice of Approximated Consistency for Knapsack Constraints. In *Proceedings of AAAI*, pages 179–184. AAAI Press, 2004.
17. M. Sellmann and T. Fahle. Constraint Programming Based Lagrangian Relaxation for the Automatic Recording Problem. *Annals of Operations Research*, 118(1):17–33, 2003.
18. M. Sellmann, G. Kliewer, and A. Koberstein. Lagrangian Cardinality Cuts and Variable Fixing for Capacitated Network Design. In *Proceedings of ESA*, pages 845–858, 2002.
19. M.A. Trick. A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. *Annals of Operations Research*, 118(1):73–84, 2003.