

# On Decomposing Knapsack Constraints for Length-Lex Bounds Consistency\*

Meinolf Sellmann

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

**Abstract.** The length-lex representation for set variables orders all subsets of a given universe of values according to cardinality and lexicography. To achieve length-lex bounds consistency for Knapsack constraints it has been proposed to decompose the constraint into two sum constraints. We provide theoretical and practical evidence which shows that decomposition increases the problem of computing a fixpoint which is intrinsic to the length-lex representation: 1. The fixpoint problem for this domain representation is NP-hard in general. 2. For a tractable sub-family of Knapsack decomposition takes more time than exponential brute-force enumeration. 3. Experimental results on decomposed Knapsack constraints show that exponential-time fixpoint computation is the rule and not some pathological exception.

## 1 Introduction

In CP, finite variable domains are commonly given by explicit enumeration of values. Then, when constraints remove values from variables' domains, a fixpoint is trivially reached after a linear number of calls to each filtering routine. The research focus in CP is therefore often concentrated on efficient constraint filtering algorithms. The situation changes fundamentally when domains become exponentially large in the instance input size, as it is, e.g., the case for set variables.

The traditional subset-superset representation for set variables elegantly circumvents the problem by ensuring that the number of potential domain changes is limited by a linear number of steps before the set variable is bound. This is however not the case for the newly proposed length-lex representation for set variables [3]. Consequently, the time until a fixpoint is reached cannot be ignored anymore.

The frequently prohibitively long time before a fixpoint is reached is a well-known practical problem in interval propagation where domains generally also have exponential size. In [1] it was rigorously proven that slow convergence is intrinsic to propagation methods. That is, the decomposition of a problem into individual constraints who exchange information only via their domains can cause NP-hard fixpoint generation problems when domains are exponentially large.

In light of these results, we study the recent proposition to decompose Knapsack constraints to achieve length-lex bounds consistency. We prove a series of negative results. 1. Computing fixpoints at which constraints are length-lex bounds consistent is NP-hard in general. 2. Constraint decomposition may introduce exponential runtimes

---

\* This work was supported by the National Science Foundation through the Career: Cornflower Project (award number 0644113).

even when the global problem is in  $P$ . 3. Experimental results on number partitioning and market split problems show that slow convergence is the rule and not some rare pathological exception.

## 2 Length-Lex Bounds Consistency

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’ and an ‘upper bound’ 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  $M(S)$  represents all mandatory elements, while the upper bound  $P(S)$  represents all possible elements, i.e.,  $D_{SS}(S) := \{s \mid M(S) \subseteq s \subseteq P(S)\}$ . We refer to this representation as the *subset* representation.

An alternate representation is based on the *length-lexicographic* ordering of the universe [3] 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.,  $D(S) := D_{LL}(S) := [L(S), U(S)] := \{s \mid L(S) \leq_{LL} s \leq_{LL} U(S)\}$ . Here  $\leq_{LL}$  denotes the length-lexicographic order which sorts sets first according to their cardinality and, as a sub-criterion when the cardinalities are equal, according to their lexicography, whereby we assume that an ordering on the universe is given and elements in a set are sorted accordingly. We refer to this representation as the *length-lex* representation. To keep the presentation simple, throughout the paper we will assume that the length-lex lower bound contains the first value and the upper bound does not as this value could otherwise be removed from consideration (see the length-lex\* representation which treats all mandatory and impossible elements separately [4]).

**Example 1.** Let  $S$  be a set-variable over the universe  $\{1, \dots, 4\}$  with domain  $D(S) = [\{1, 3\}, \{2, 3, 4\}]$ . Then, the values that  $S$  can take are  $D(S) = \{\{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}\}$ .

For constraints involving set-variables, the filtering task is to tighten the bounds of the domains. The respective definition of bounds consistency depends on the domain representation used. For the traditional subset representation, bounds consistency of a unary constraint over a set variable means that all values that must be part of the final set according to the constraint are included in the lower bound and all values that cannot be part of the final set are not elements of the upper bound. For the length-lex representation, bounds consistency simply means that both the lower and the upper bound are consistent according to the constraint.

This latter property appears highly desirable. However, as we will see shortly, it is the reason why computing fixpoints of length-lex bounds consistency is intractable.

## 3 Decomposing Knapsack Constraints

Consider the *Knapsack constraint*  $KP(S, p, B, w, C)$  where we are given a set of items modeled by a set-variable  $S$  over a universe of  $n$  items, a profit vector  $p$  which gives

the non-negative profit of each item, a weight vector  $w$  which gives the weight of each item, a lower bound  $B$  on the total profit, and a capacity  $C$  bounding the total weight of the knapsack. The constraint requires that the set  $s$  assigned to  $S$  fulfills  $\sum_{i \in S} p_i \geq B$ , and  $\sum_{i \in S} w_i \leq C$ . In [4] we studied the question whether Knapsack problems are still approximable when conjoined with cardinality and lexicographic constraints. We proved that this is possible and gave an fully polynomial time approximation scheme for the problem. Based on this scheme, we proved that approximated length-lex bounds consistency can be achieved in polynomial time.

While the runtime of our filtering algorithm is polynomial, it is too expensive to be invoked in a constraint propagation engine. Therefore, as a practical alternative to this algorithm, in [7] it is suggested to decompose  $KP$  into one sum constraint which enforces the capacity bound and another sum constraint which enforces the profit bound. A complex algorithm for these constraints is devised which requires a preprocessing step in  $O(n^3)$  and then runs in amortized time  $O(n \log n)$  for each sum constraint. We will now analyze the proposed decomposition theoretically and practically.

### 3.1 Knapsack Decomposition in Theory

We begin by stating a trivial intractability result which shows that slow-convergence when generating fixpoints is an intrinsic problem of length-lex bounds consistency.

**Lemma 1.** *It is NP-hard to decide whether a fixpoint exists to a system of only one set-variable and just two unary constraints with associated monotonic and idempotent filtering operators that achieve length-lex bounds consistency.*

*Proof.* We reduce from Knapsack, i.e., we want to decide whether there exists a subset  $I \subseteq \{1, \dots, n\}$  such that setting  $S \leftarrow I$  satisfies constraint  $W(S) := \sum_{i \in S} w_i \leq C$  and constraint  $P(S) := \sum_{i \in S} p_i \geq B$  for natural numbers  $w_i, p_i, C$ , and  $B$ . Note that  $W$  and  $P$  both affect both bounds of the length-lex domain. [7] devised monotonic and idempotent filtering operators for the constraints  $W(S)$  and  $P(S)$ . If there exists a fixpoint where both constraints are length-lex bounds consistent, then, at the bounds found, both constraints are satisfied. Therefore, each bound gives a solution that satisfies both constraints, and thus solves the Knapsack problem. On the other hand, assume that the Knapsack instance is valid, i.e., there exists an  $I \subseteq \{1, \dots, n\}$  such that  $W(S)$  and  $P(S)$  are satisfied when setting  $S \leftarrow I$ . Then,  $D(S) = [I, I]$  is a fixpoint where both constraints are length-lex bounds consistent.  $\square$

**Remark 1.** The proof above shows that the problem with fixpoint intractability is not actually caused by the specific (in our case: length-lex) ordering of the set variable's domain. The same argument can be made for *any* form of set bounds consistency which requires that constraints are satisfied at the domain bounds.

While the intractability of computing a fixpoint is certainly disconcerting, it is not really surprising since we are tackling an NP-hard problem after all. However, even more concerning is the following:

**Lemma 2.** *Decomposing Knapsack constraints into two sum constraints in a propagation engine can introduce exponential runtimes even for tractable problems.*

*Proof.* Consider the following family of Knapsack instances which belong to the large class of Knapsack problems in  $P$  for which  $\min\{\|p\|_\infty, \|w\|_\infty\}$  is bounded by a polynomial in  $n$ . For any even number of variables  $n$  we set  $p_1 = w_1 = n, p_n = w_n = n$ , and  $p_i = w_i = 1$  for all  $1 < i < n$ . Assume we introduce a set variable  $S \subseteq \{1, \dots, n\}$  and pose two separate constraints,  $W(S)$  which enforces  $\sum_{i \in S} w_i \leq 3n/2$ , and  $P(S)$  which enforces  $\sum_{i \in S} p_i \geq 3n/2$ .

Observe how the lower length-lex bound evolves during the fixpoint computation: After filtering  $W$  and  $P$  twice, we observe the sequence  $\{1, 3, 4\}, \{1, 3, n\}, \{1, 4, 5\}, \{1, 4, n\}, \dots, \{1, n-2, n\}$ . That is, we observe *all* sets of cardinality 3 which contain 1 and  $n$  and do not contain  $n-1$ .

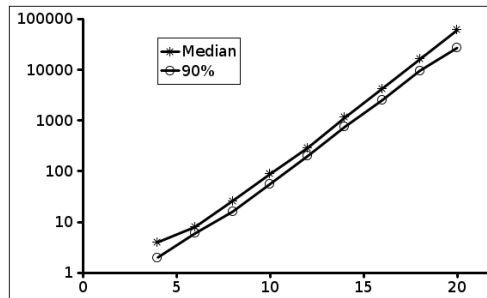
The fixpoint computation continues:  $\{1, 2, 3, 4\}, \{1, 2, 3, n\}, \{1, 2, 4, 5\}, \{1, 2, 4, n\}, \dots, \{1, 2, n-2, n\}, \{1, 3, 4, 5\}, \{1, 3, 4, n\}, \{1, 3, 5, 6\}, \{1, 3, 5, n\}, \dots, \{1, 3, n-2, n\}, \dots, \{1, n-3, n-2, n\}$ . That is, among others we visit *all* sets of cardinality 4 which contain 1 and  $n$  and do not contain  $n-1$ .

Next we visit, among other sets, *all* sets of cardinality 5 which contain 1 and  $n$  and do not contain  $n-1$ . And so on and so forth until we finally find the lower bound of cardinality  $n/2 + 1, \{1, 2, \dots, n/2 + 1\}$ . We observe: by filtering constraints  $W$  and  $P$  separately and in turn, we implicitly compute a sequence of increasing lower bounds in the length-lex representation of all sets of cardinality from 3 to  $n/2$  which contain items 1 and  $n$  and not  $n-1$ . Consequently, for  $n \geq 6$ , the fixpoint computation requires more than  $\sum_{i < n/2-1} \binom{n-3}{i} \geq 2^{n-5}$  calls to the filtering methods, and thus runs in time  $\Omega(2^n n \log n)$  – more than brute force enumeration. On the other hand, the pseudo-polynomial approach from [4] can be followed to compute the same bounds in time  $O(n^3)$ .  $\square$

### 3.2 Knapsack Decomposition in Practice

**Number Partitioning.** The previous discussion shows that Knapsack decomposition for length-lex bounds consistency is certainly not appealing from a theoretical worst-case point of view. The question arises whether slow convergence occurs only rarely in practice. One of the simplest problems for which we may consider employing a filtering algorithm for Knapsack constraints is number partitioning. Given numbers  $a_1, \dots, a_n$ , is there a set  $I \subseteq \{1, \dots, n\}$  such that  $\sum_{i \in I} a_i = \frac{\sum_i a_i}{2}$ ?

Figure 1 shows the number of filtering calls of 100 small, randomly generated number partitioning problems modeled by one set variable with one unary Knapsack constraint that is decomposed into two sum constraints. The two curves (logarithmic scale) depict lower bounds on



**Fig. 1.** Filtering calls (log-scale) on number partitioning problems with 4 to 20 numbers (100 instances per data point, numbers drawn uniformly from  $[1, 10^6]$ ). The upper curve shows the minimum number of calls needed by at least 50% of the instances, the lower the minimum number of calls needed by at least 90% of the instances.

the number of calls that are required until a fixpoint is reached for 90% and 50% of the instances. We see clearly that exponential-time fixpoint computations occur in the vast majority of test cases. Slow convergence is thus the predominant rule and not some pathological worst-case exception.

**Speeding up Convergence.** In order to show that Knapsack decomposition is not appealing in practice either, in the following we introduce a heuristic improvement which tries to address the slow convergence problem. We will then show that even this improved algorithm cannot compete with non-decomposition approaches.

To speed up convergence we exploit the method based on dynamic programming (DP) from [4] which simplifies considerably for sum constraints. Let us consider sum constraints of the form  $\sum_{i \in I} w_i \leq C$ , with  $w_i, C \in \mathbb{Q}$ . We set up a dynamic program in two dimensions (compare with Figure 2). The first gives the current index of the largest element allowed in the solution set, which is restricted to be a subset of  $\emptyset, \{1\}, \{1, 2\}, \dots, \{1, \dots, n\}$ . The other dimension gives the cardinality  $0, 1, \dots, n$ . Then, in  $W_{ic}$  we store the weight of the set  $I \subseteq \{1, \dots, i\}$  with cardinality  $|I| = c$  that has the smallest weight  $w(I) := \sum_{i \in I} w_i$ . The ordinary recursion equation  $W_{i+1,c} \leftarrow \min\{W_{ic}, W_{i-1,c-1} + w_i\}$  can be used to find the set  $I \subseteq \{1, \dots, n\}$  with cardinality  $\kappa$  that minimizes  $w(I) = W_{n\kappa}$ .

For the purpose of achieving length-lex bounds consistency for sum constraints, this basic DP can be adapted following the same ideas as given in [4]. Assume that we are given a set variable  $S \subseteq \{1, \dots, n\}$  with domain  $D(S) = [L, U]$ ,  $L, U \subseteq \{1, \dots, n\}$ . We are to find new lower and upper bounds  $L', U' \subseteq \{1, \dots, n\}$  such that  $w(L'), w(U') \leq C$  and for all  $L'', U''$  with  $L \leq_{LL} L'' <_{LL} L', U' <_{LL} U'' \leq_{LL} U$  it holds  $w(L''), w(U'') > C$ . Following the approach in [4], it is sufficient to devise an algorithm for the case where  $\kappa \leftarrow |L| = |U|$ .

Note that  $L$  and  $U$  define paths  $\pi_L$  and  $\pi_U$  in the DP from  $W_{00}$  to  $W_{n\kappa}$  (see Figure 2). Analogously, every path  $\pi$  from  $W_{00}$  to  $W_{n\kappa}$  defines a set  $S_\pi \subseteq \{1, \dots, n\}$ . We call a path *admissible* iff  $L \leq_{LL} S_\pi \leq_{LL} U$ . It will be important for us to know the shortest path distance from the root to a given node when the choices implied by that path  $\pi$  already ensure that the resulting set  $S_\pi$  must obey the lexicographic bounds  $L, U$ . Conversely, we will also need to argue about paths from nodes in the DP-induced graph to the sink-node  $W_{n\kappa}$ :

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

For a path  $\pi$  from  $W_{ic}$  to  $W_{n\kappa}$ , we write  $L \leq_{lex} S_\pi$  (or  $S_\pi \leq_{lex} U$ ) if and only if for  $T \leftarrow S_\pi \cup (L \cap \{1, \dots, i\})$  ( $T \leftarrow S_\pi \cup (U \cap \{1, \dots, i\})$ ) it holds that  $|T| = \kappa$  and  $L \leq_{lex} T$  ( $T \leq_{lex} U$ ).

Now, for every node  $W_{ic}$  in the DP, we compute the following quantities:

1. For  $W_{ic} \in \pi_L$ ,  $M_{ic}^1$  gives the distance from the root  $W_{00}$  to  $W_{ic}$  when following  $\pi_L$ , that is  $M_{ic}^1 \leftarrow \sum_{i \in L, i \leq k} w_i$ . For  $W_{ic} \notin \pi_L$ , we set  $M_{ic}^1 \leftarrow \infty$ .
2. For  $W_{ic} \in \pi_U$ ,  $M_{ic}^2$  gives the distance from the root  $W_{00}$  to  $W_{ic}$  when following  $\pi_U$ , that is  $M_{ic}^2 \leftarrow \sum_{i \in U, i \leq k} w_i$ . For  $W_{ic} \notin \pi_U$ , we set  $M_{ic}^2 \leftarrow \infty$ .

3. For arbitrary nodes  $W_{ic}$ ,  $M_{ic}^3$  gives the length of the shortest path  $\pi$  from the root to  $W_{ic}$  with  $L <_{lex} S_\pi <_{lex} U$ .
4. For arbitrary nodes  $W_{ic}$ ,  $M_{ic}^4$  gives the length of the shortest path  $\pi$  from  $W_{ic}$  to  $W_{n\kappa}$ .
5. For  $W_{ic} \in \pi_L$ ,  $M_{ic}^5$  gives the length of the shortest path  $\pi$  from  $W_{ic}$  to  $W_{n\kappa}$  with  $L \leq_{lex} S_\pi$ .
6. For  $W_{ic} \in \pi_U$ ,  $M_{ic}^6$  gives the length of the shortest path  $\pi$  from  $W_{ic}$  to  $W_{n\kappa}$  with  $S_\pi \leq_{lex} U$ .

In [4] we devised recursion equations that allow us to compute all these values in time  $O(n^2)$ . Based on this data, we can compute the desired bounds  $L', U'$ . To this end, we propose a new procedure which is more efficient than the one presented in [4].

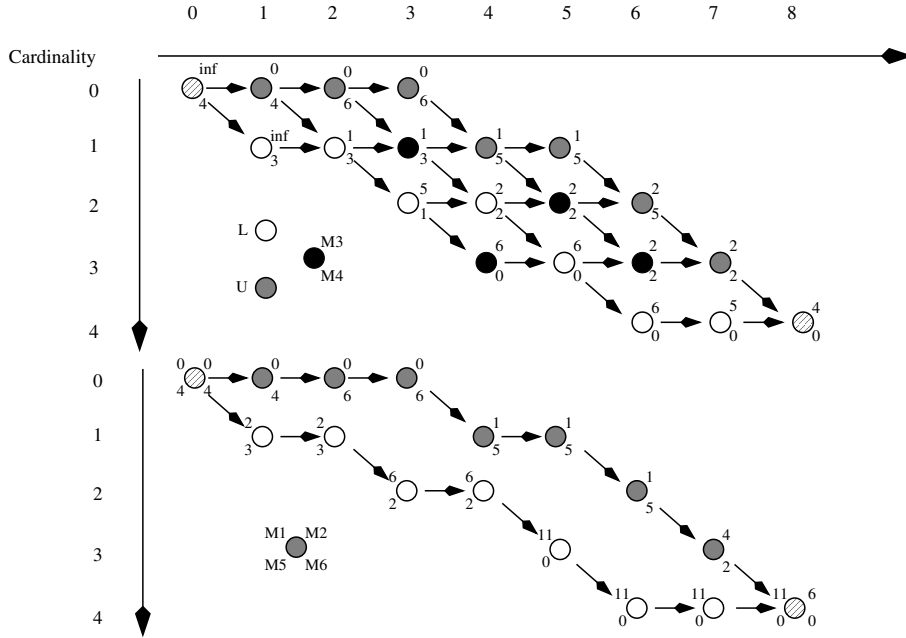
Let us consider the problem of computing  $L'$  (the computation of  $U'$  works analogously). We start at node  $W_{00}$ . Recall that we may assume that  $1 \in L$  and  $1 \notin U$ . We start at node  $W_{00}$  and initialize the already committed cost  $g \leftarrow 0$ . Clearly  $1 \in L'$  if and only if  $g + w_1 + M_{11}^5 \leq C$ . In this case, we move to  $W_{11}$  and update  $g \leftarrow g + w_1$ . While we stay on  $\pi_L$ , we update in this way: at  $W_{ic}$ , if  $i + 1 \notin L$  or when  $g + w_{i+1} + M_{i+1,c+1}^5 > C$ , we move to  $W_{i+1,c}$ . Otherwise we add  $i + 1$  to  $L'$ , move to  $W_{i+1,c+1}$ , and update  $g \leftarrow g + w_{i+1}$ . At some point it may occur that  $i + 1 \in L$  but  $g + w_{i+1} + M_{i+1,c+1}^5 > C$ . Then, we leave  $\pi_L$  and we are already guaranteed that  $L <_{LL} L'$ . So we continue as follows: If  $g + w_{i+1} + M_{i+1,c+1}^4 \leq C$ , we add  $i + 1$  to  $L'$ , move to  $W_{i+1,c+1}$ , and update  $g \leftarrow g + w_{i+1}$ . Otherwise, we move to  $W_{i+1,c}$ . In this way, we can compute the new lower bound  $L'$  in time  $O(n)$  once the values  $M_{ic}^k$  are known. With this procedure, length-lex bounds consistency for sum constraints can be achieved in time  $O(n^2)$ .

The previous algorithm is very simple and easy to implement. In contrast to the algorithm presented in [7], it requires no specialized theory, cubic preprocessing, or amortized runtime analysis. On the other hand, it is slower than the  $O(n \log n)$  amortized time from [7]. However, as we have seen earlier, the problem with decomposing Knapsack constraints is *not* caused by the inefficiencies when propagating sum constraints. The problem is the very *slow convergence* before a fixpoint is reached. The algorithm presented above has the potential to reduce this time by filtering *all* sum constraints within the *same* dynamic program.

To achieve this practical improvement, we will filter edges from the graph induced by the DP. Since a sum constraint is a Knapsack constraint where the profits are all zero, from Lemma 1 in [4], we know that the following quantities can be used to remove edges from the DP-induced graph:

### Lemma 3

- The length of a shortest admissible path through an edge  $(W_{ic}, W_{i+1,c})$  is  $\min\{M_{ic}^3 + M_{i+1,c}^4, M_{ic}^1 + D_{ic}^1, M_{ic}^2 + D_{ic}^2\}$ , where  $D_{ic}^1 = M_{i+1,c}^4$  if  $i + 1 \in L$  and  $D_{ic}^1 = M_{i+1,c}^5$  otherwise, and  $D_{ic}^2 = M_{i+1,c}^6$  if  $i + 1 \notin U$  and  $D_{ic}^2 = \infty$  otherwise.
- The length of a shortest admissible path through an edge  $(W_{ic}, W_{i+1,c+1})$  is  $\min\{M_{ic}^3 + M_{i+1,c+1}^4 + w_{i+1}, M_{ic}^1 + E_{ic}^1 + w_{i+1}, M_{ic}^2 + E_{ic}^2 + w_{i+1}\}$ , where  $E_{ic}^1 = M_{i+1,c+1}^5$  if  $i + 1 \in L$  and  $E_{ic}^1 = \infty$  otherwise, and  $E_{ic}^2 = M_{i+1,c+1}^4$  if  $i + 1 \notin U$  and  $E_{ic}^2 = M_{i+1,c+1}^6$  otherwise.



**Fig. 2.** The DP for the weight constraint in Example 1. The top gives values  $M^3$  and  $M^4$  for all nodes, the bottom  $M^1, M^2, M^5, M^6$  for nodes on  $L$  and  $U$ . Note that node  $W_{4,3}$  is valid as  $\{2, 3, 4, 6\}$  obeys the length-lex bounds and defines a path that visits this node. Edge  $(W_{6,3}, W_{7,4})$  has been removed previously by the profit constraint.

**Example 2.** For a set variable  $S \subseteq \{1, \dots, 8\}$ , consider the Knapsack constraint  $KP(S, (0, 0, 0, 0, 0, 0, -1, 1)^T, 0, (2, 1, 4, 1, 5, 0, 3, 2)^T, 7)$ . That is, we want a subset of items  $S$  such that  $x_8 - x_7 \geq 0$  and  $2x_1 + x_2 + 4x_3 + x_4 + 5x_5 + 3x_7 + 2x_8 \leq 7$ , where  $x_i = 1$  if  $i \in S$  and  $x_i = 0$  otherwise. Assume further that  $D(S) = [\{1, 3, 5, 6\}, \{4, 6, 7, 8\}]$ . Propagating the profit constraint  $x_7 - x_8 \leq 0$ , we detect that the edge  $(W_{6,3}, W_{7,4})$  cannot be used by any admissible path with weight lower or equal 0. Then, we propagate the constraint  $2x_1 + x_2 + 4x_3 + x_4 + 5x_5 + 3x_7 + 2x_8 \leq 7$ . Figure 2 shows the DP, the lower and upper bounds, as well as all values  $M_{ic}^k$  at this point. To compute a new lower bound, we initialize  $g \leftarrow 0$  and begin at  $W_{0,0}$ . It holds  $g + w_1 + M_{11}^5 = 0 + 2 + 3 \leq 7$ , so we include 1 in  $L'$  and update  $g \leftarrow 2$ . Since we are still on  $L$  and  $2 \notin L$ , we cannot include 2 in  $L'$  and move to  $W_{2,1}$ . Then,  $2 + 4 + 2 > 7$ , so we move to  $W_{3,1}$ , leaving  $\pi_L$ . From now on we will use  $M^4$  instead of  $M^5$ . Next,  $g + w_4 + M_{4,2}^4 = 2 + 1 + 2 \leq 7$ , so we include 4 in  $L'$ , update  $g \leftarrow 3$ , and move to  $W_{4,2}$ . Then,  $3 + 5 + 0 > 7$ , so we move to  $W_{5,2}$ . Next,  $3 + 0 + 2 \leq 7$ , so we include 6 in  $L'$ , update  $g \leftarrow 3$ , and move to  $W_{6,3}$ . Now, the edge  $(W_{6,3}, W_{7,4})$  has been removed, so we move to  $W_{7,3}$ . Finally, we find  $3 + 2 + 0 \leq 7$  and include 8 in  $L'$ . The new lower bound is  $L' = \{1, 4, 6, 8\}$ . Note that, without the prior removal of  $(W_{6,3}, W_{7,4})$ , the new bound would have been  $\{1, 4, 6, 7\}$ . This bound would disagree with the profit

**Table 1.** Test results on 100 random market split problems. We give number of variables, constraints, the average number of filtering calls, time in CPU seconds, and, in case where the number of pure bound computations is limited, the average number of choice points.

Vars	Cons	LL		LL-red		LL-red-100			LL-red-100-noEdge		
		Filt. Calls	Time	Filt. Calls	Time	CPs	Filt. Calls	Time	CPs	Filt. Calls	Time
10	2	55.1	0	47.0	0	1	47.0	0	1	109	0
20	3	34.8K	0.58	34.0K	0.57	9.94	31.3K	0.49	31.5	35.0K	0.62
30	4	14.4M	481	14.3M	475	2.9K	12.8M	298	9.3K	13.4M	426
40	5	> 500M	> 5h	> 500M	> 5h	> 100K	> 500M	> 5h	> 100K	> 500M	> 5h

constraint  $x_7 - x_8 \leq 0$ , resulting in a need to update the bound again by the profit constraint. The removal of edges and the propagation of sum constraints within the same DP graph can thus contribute to faster convergence in practice.

We test this algorithm on Market Split Problems which consist in the conjunction of several number partitioning constraints. We use randomly generated Cornuejols/Dawande instances [2] for our tests. Our length-lex approach - which does not need to branch as a fixpoint of length-lex bounds consistency satisfies all constraints, is referred to as “LL”. To strengthen the filtering power, we use redundant constraints which result from a weighted sum of the original equalities as they were introduced in [6] and later used successfully in [5]. The respective approach is called “LL-red”. In practice it may be beneficial not to wait until a fixpoint of length-lex bounds consistency has been achieved. We test a third approach where we continue propagating sum constraints within the same DP until for 100 iterations only the length-lex bounds have changed, but no edge could be removed. In this case, branching becomes of course necessary. We refer to this approach as “LL-red-100”. Finally, we experiment with the latter approach where we turn off the edge-removal heuristic and to which we refer as “LL-red-100-noEdge”. Tests were run on an AMD Athlon 64 3800+ 2.0GHz CPU.

We see that adding redundant constraints helps a little, and limiting the number of pure length-lex bounds improvements helps further reduce the total number of filtering calls. Moreover, our edge-removal heuristic effectively reduces the number of filtering calls, the number of choice points, and the total time needed. However, even with all these improvements the decomposition approach is not competitive at all. The non-decomposition algorithm based on approximated subset-bounds consistency from [5], run on an AMD Athlon 1.2GHz, solves on average instances with 4 constraints in less than a second while visiting less than 6 choice points, and instances with 5 constraints in less than a minute while visiting less than 60 choice points.

## 4 Conclusion

We rejected the conjecture from [7] that vanilla Knapsack constraint decomposition was appealing for achieving length-lex bounds consistency. In theory, it can lead to exponential filtering times even for tractable filtering problems. In practice, even when using improvements like propagating sum constraints within the same DP, it is vastly outperformed by an approach based on non-decomposed Knapsack constraint filtering.

In general, we argue that filtering algorithms achieving length-lex bounds consistency must not be evaluated based on the filtering-time alone but always have to be viewed as posing a fixpoint problem that is potentially hard in theory and in practice. Decomposing constraints for achieving length-lex bounds consistency appears particularly disadvantageous as this puts even more burden on the fixpoint algorithm.

## References

1. Bordeaux, L., Hamadi, Y., Vardi, M.Y.: An Analysis of Slow Convergence in Interval Propagation. In: CP, pp. 790–797 (2007)
2. Cornuejols, G., Dawande, M.: A Class of Hard Small 0-1 Program. In: IPCO, pp. 284–293 (1998)
3. Gervet, C., Van Hentenryck, P.: Length-lex ordering for set CSPs. In: AAI (2006)
4. Malitsky, Y., Sellmann, M., van Hoeve, W.-J.: Length-Lex Bounds Consistency for Knapsack Constraints. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 266–281. Springer, Heidelberg (2008)
5. Sellmann, M.: The Practice of Approximated Consistency for Knapsack Constraints. In: AAI, pp. 179–184 (2004)
6. Trick, M.: A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. In: CPAIOR, pp. 113–124 (2001)
7. Yip, J., Van Hentenryck, P.: Length-Lex Bound Consistency for Knapsack Constraints. In: ACM SAC (2009)