

Length-Lex Bound Consistency for Knapsack Constraints

Justin Yip
Brown University
Box 1910, Providence
RI02912, USA
justin@cs.brown.edu

Pascal Van Hentenryck
Brown University
Box 1910, Providence
RI02912, USA
pvh@cs.brown.edu

ABSTRACT

This paper considers the length-lex domain for set-variables in constraint programming which includes not only membership but also cardinality and lexicographic information. The paper studies how to enforce bound consistency on a knapsack constraint over a set variable in this domain and proposes a bound-consistency algorithm which runs in $O(c \log n)$ time with a one-time preprocessing cost $O(cn^2)$ when the constraint is posted.

Categories and Subject Descriptors

D.3.2 [Language Classifications]: Constraint and logic languages;

D.3.3 [Language Constructs and Features]: Constraints

Keywords

Set variable, Length-lex, Bound Consistency, Knapsack

1. INTRODUCTION

The length-lex representation has been recently proposed for representing sets in CSPs [4] and it offers two computational benefits. First, contrary to earlier set representations such as the subset-bound domain [5, 3], it features a total ordering on sets, which makes it possible to define, and enforce, bound consistency. Second, the length-lex domain directly integrates cardinality and lexicographic information which are so important in set CSPs [1, 6].

Gervet and Van Hentenryck [4] showed how to enforce bound consistency on several unary constraints (e.g., inclusion/exclusion) in time $O(c)$, where c is the cardinality of the set variable. Recently, Van Hentenryck, Yip, Gervet, and Dooms [7] presented a generic bound-consistency algorithm for arbitrary binary constraints over sets. The bound-consistency algorithm only assumes the existence of an algorithm to check whether a constraint C has a solution in two length-lex intervals X and Y , i.e., $\exists s \in X, t \in Y : C(s, t)$. and makes $O(c^2 \log n)$ calls to the feasibility algorithm, where c is the cardinality of the sets and n is the size of the universe, i.e., the set of elements which may appear in the sets.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.

Copyright 2009 ACM 978-1-60558-166-8/09/03 ...\$5.00.

This paper considers the knapsack constraint

$$C(X) \equiv \sum_{e \in X} w(e) \leq b.$$

where X is a length-lex set variable ranging over U , $w : U \rightarrow \mathcal{R}$, and a scalar $b \in \mathcal{R}$. It shows how to enforce bound consistency on knapsack constraints in $O(c \log n)$ time with a one-time $O(cn^2)$ preprocessing time when the constraint is posted. This is an interesting result, since the knapsack constraint is a building block of many applications in which set variables occur including configurations and network design problems and the length-lex domain directly incorporates cardinality and lexicographic which also arises naturally in these settings. The result also provides the basis for an appealing alternative to the $O(\frac{n^4}{\epsilon})$ algorithm of [2] for enforcing approximated bound consistency on the conjunction of two knapsack constraints over the same length-lex set variable.

The rest of this paper is organized as follows. Section 2 gives the background on the length-lex domains and specifies the problem. Section 3 presents a propagation algorithm based on the decomposition scheme of [7] that runs in time $O(c^2 + c \log n)$ time. Section 4 shows how to reduce the complexity to $O(c \log n)$ by amortizing the decomposition. Section 5 shows how to apply the results to other constraints and discusses related work. Section 6 concludes the paper.

2. BACKGROUND AND DEFINITIONS

This section reviews the main concepts behind the length-lex domain [4, 7] and specifies the knapsack constraints.

For simplicity, we assume that sets take their values in a universe U of integers $\{1, \dots, n\}$ equipped with traditional set operations. Elements of U are denoted by the letters f, l and h possibly subscripted and sets are denoted by the letters m, M, s , and x . A subset m of U of cardinality c is denoted $\{m_1, m_2, \dots, m_c\}$ ($m_1 < m_2 < m_3 \dots < m_c$) and thus m_j denotes the j -th smallest value in m . The notation $m_{i..j}$ is the shorthand for $\{m_i, m_{i+1}, \dots, m_j\}$. Finally, we call c -set any set of cardinality c . Algorithms in this paper are only given for a fixed cardinality c but are easily extended to the general case.

The length-lex ordering \preceq proposed in [4], totally orders sets first by cardinality and then lexicographically.

DEFINITION 1. *The length-lex ordering \preceq is defined by:*

$$s \preceq t \text{ iff } s = \emptyset \vee |s| < |t| \vee$$

$$|s| = |t| \wedge (s_1 < t_1 \vee s_1 = t_1 \wedge s \setminus \{s_1\} \preceq t \setminus \{t_1\})$$

EXAMPLE 1. *Given $U = \{1, \dots, 4\}$, we have $\emptyset \preceq \{1\} \preceq \{2\} \preceq \{3\} \preceq \{4\} \preceq \{1, 2\} \preceq \{1, 3\} \preceq \{1, 4\} \preceq \{2, 3\} \preceq \{2, 4\} \preceq \{3, 4\} \preceq \{1, 2, 3\} \preceq \{1, 2, 4\} \preceq \{1, 3, 4\} \preceq \{2, 3, 4\} \preceq \{1, 2, 3, 4\}$.*

DEFINITION 2. Given a universe U , a length-lex interval is a pair of sets $\langle m, M \rangle$. It represents the sets between m and M in the length-lex ordering, i.e., $\{s \subseteq U \mid m \preceq s \preceq M\}$.

EXAMPLE 2 (LENGTH-LEX INTERVAL). Given $U = \{1, \dots, 6\}$, the interval $\langle \{1, 3, 4\}, \{1, 5, 6\} \rangle$ denotes the set $\{\{1, 3, 4\}, \{1, 3, 5\}, \{1, 3, 6\}, \{1, 4, 5\}, \{1, 4, 6\}, \{1, 5, 6\}\}$.

Because the length-lex ordering is a total order on sets, it is possible to enforce bound consistency on set constraints. A unary constraint is bound-consistent with respect to a length-lex interval if both bounds of the intervals belong to the solution.

DEFINITION 3 (BOUND CONSISTENCY). A unary constraint C over set variable S with domain $X = \langle m, M \rangle$ is bound-consistent if $C(m) \wedge C(M)$.

A knapsack constraint bounds the sum of the weights of the elements of a set variable.

DEFINITION 4 (KNAPSACK CONSTRAINT). A knapsack constraint $kp(w, b)$ receives as input a function $w : U \rightarrow \mathcal{R}$ and a scalar $b \in \mathcal{R}$. It holds for a set s (denoted by $kp(w, b)(s)$) if

$$\sum_{e \in s} w(e) \leq b.$$

This paper is to present efficient algorithms enforcing bound consistency on the knapsack constraint in the length-lex domain.

Reference [7] proposed a special class of length-lex interval, called PF-closed interval, that enjoys some nice closure properties. Intuitively, PF-closed interval is defined by four parameters P , F , V and c and it denotes all the c -sets that starts with a common prefix P , filled by elements from V , and containing at least one element in F . Notice that each element in F is always smaller than those elements in $V \setminus F$ and that $F \subseteq V$.

DEFINITION 5 (PF-CLOSED INTERVALS). Let P , F , and V be sets and c be an integer. A PF-closed interval $pf\langle P, F, V, c \rangle$ satisfies

$$\max(P) < \min(F) \wedge F \subseteq V \wedge \forall e \in V \setminus F : e > \max(F) \wedge |V \setminus F| \geq c - |P| - 1$$

and denotes the set of sets

$$\{P \cup \{f\} \cup s \mid f \in F \wedge s \subseteq V \wedge |P \cup \{f\} \cup s| = c\}.$$

Any length-lex interval of c -sets can be partitioned into $O(c)$ PF-closed intervals, which is fundamental computationally.

LEMMA 1. A length-lex interval of c -sets can be decomposed into $O(c)$ PF-closed intervals in $O(c^2)$ time.

The details of the decomposition and the proof of these results can be found in [7]. Note also that, although the definition of PF-closed intervals does not impose F and V to be ranges, the decomposition algorithm *decomp* always produces PF-closed intervals in which both sets are ranges. Hence, in the rest of this paper, we assume that F and V are ranges, which allows us to obtain more efficient algorithms. We also denote $F = \{F_m, \dots, F_M\}$ and $V = \{F_m, \dots, n\}$.

3. BOUND CONSISTENCY

This section provides a simple algorithm to enforce bound consistency for knapsack constraints in $O(c^2 \log n)$ with a one-time $O(cn^2)$ preprocessing time. This algorithm is very similar to the

i	1	2	3	4	5	6	7	8
$w(i)$	2	1	4	1	5	0	3	2

Table 1: The Weight Function for the Examples

structure of the generic algorithm that enforces bound consistency for binary constraints in [7]. Indeed, to achieve bound consistency, we first decompose a length-lex interval into a list of PF-closed intervals, we then search for the minimum that contains an element satisfying the constraint, and we finally construct a minimum set in that interval that satisfies the constraint. This section gives the feasibility algorithm for a PF-closed interval and the construction algorithm for finding the first feasible successor wrt a knapsack constraint. The algorithm for finding the first feasible predecessor is similar and is thus omitted.

Unless otherwise stated, we use the following settings for the illustrations throughout the paper: a universe U with 8 items ($U = \{1, \dots, 8\}$), a length-lex interval $\langle \{1, 3, 5, 6\}, \{4, 6, 7, 8\} \rangle$, and a knapsack constraint $kp(w, b)$ where w is a weight function shown in Table 1 and $b = 7$. We use a shorthand $w(s)$ (where s is a set) to denote the sum of weight of all items in s .

3.1 Feasibility Check

The feasibility check $hs(C)$ determines whether there is a solution in length-lex intervals wrt a constraint C . For unary constraints, it can be specified as follows.

SPECIFICATION 1 (HASSOLUTION). Given a unary constraint C and a length-lex interval X , $hs(C)(X) \equiv \exists x \in X : C(x)$.

This subsection gives the feasibility routine for a PF-closed interval i.e., $hs(kp(w, b))(X_{pf} = pf\langle P, F, V, c \rangle)$. Since the prefix P is fixed, we subtract $w(P)$ from b to obtain the remaining capacity b' . The main computational problem is thus to determine whether there exists a set s in the F-closed interval $f\langle F, V, c - |P| \rangle$ (F-closed interval is a special case of PF-closed interval in which $P = \emptyset$) that satisfies $kp(w, b')$. Notice that each set in this interval takes at least one element f from F and the remaining ones from V and that f is always the smallest element according to the definition. Hence the goal is to find a set $\{s_1, \dots, s_c\}$ with $s_1 \in F$. Once the smallest element s_1 is fixed, we fill the remaining positions by picking the $c - 1$ smallest item from $\{s_1 + 1, \dots, n\}$. The feasibility routine enumerates all possible choice for the smallest element. If any of those results in a set with weight not more than the upper bound b' , it returns true; otherwise, it returns false. We start with examples.

EXAMPLE 3. Let $X_{pf} = pf\langle \{1\}, \{4, \dots, 6\}, \{4, \dots, 8\}, 4 \rangle$. Since the weight of the prefix is 2, the feasibility problem now becomes $hs(kp(w, 5))(f\langle \{4, \dots, 6\}, \{4, \dots, 8\}, 3 \rangle)$. The feasibility routine returns true if and only if there exists a 3-set s in the F-closed interval with weight not more than 5. It first enumerates all the possible choices for the first element. Suppose the first element is 4: since $w(4) = 1$, we must find two elements greater than 4 whose weights sum to not more than 4. It is an easy task, since we only need to know if the sum of the 2 smallest item from 5 to 8 is not more than 4. In our example, items 6 and 8 are the 2 smallest with total weight of 2. Hence, X_{pf} has a solution wrt $kp(w, 7)$ and the feasibility routine returns true.

EXAMPLE 4. Let $X_{pf} = pf\langle \{1, 3\}, \{5, \dots, 7\}, \{5, \dots, 8\}, 4 \rangle$. The weight of the prefix is 6 and hence it leaves only 1 for the remaining two items. The weights of the two smallest items from 5 to 8 sum to 2 and hence they do not fit into the remaining capacity. Hence, there is no solution in X_{pf} wrt $kp(w, 7)$ and the feasibility routine returns false.

$c = 3$	F_M							
	1	2	3	4	5	6	7	8
1	3	2	2	2	2	2	2	2
2		2	2	2	2	2	2	2
3			5	3	3	3	3	3
F_m 4				3	3	3	3	3
5					7	5	5	5
6						5	5	5
7								
8								

Figure 1: The Lookup Table (Unfilled Cells Represent $+\infty$)

Algorithm 1 $hs(kp(w, b))(X_{pf} = pf\langle P, F, V, c \rangle)$

1: **return** $wt(F_m, F_M, c - |P|) \leq b - w(P)$

The feasibility routine $hs(kp(w, b))(X_{pf} = pf\langle P, F, V, c \rangle)$ is equivalent to the following expression:

$$\min_{s_1 \in F} w(s_1) + \min\{w(s_{2..c'}) \mid s_{2..c'} \subseteq \{s_1 + 1, \dots, n\}\} \leq b - w(P)$$

where $c' = c - |P|$. Observe that the left side of the above expression is equal to the minimum weight set in a F-closed interval. Moreover, we can dramatically improve the efficiency by using a preprocessed lookup table returning the minimum weight set in the corresponding F-closed interval.

SPECIFICATION 2 (MIN-WEIGHT-LOOKUP-TABLE). Given a range $\{F_m, \dots, F_M\}$, a cardinality c , and a weight function w , the min-weight-lookup-table wt for a universe size n is defined by a function $wt(F_m, F_M, c)$ whose value is ¹

$$\min_{s_1 \in F_m..F_M} w(s_1) + \min\{w(s_{2..c}) \mid s_{2..c} \subseteq \{s_1 + 1, \dots, n\}\}.$$

Figure 1 depicts the lookup table for $c = 3$ on our example data. Given this lookup table, it is easy to implement a feasibility algorithm for knapsack constraints. Indeed, Algorithm 1 implements Specification 1 for $\mathcal{C} = kp(w, b)$.

LEMMA 2. Given a lookup table wt , $hs(kp(w, b))(X_{pf})$ (Algorithm 1) takes $O(c)$ time.

PROOF. Computing $w(P)$ takes $O(c)$ and a table lookup takes constant time. \square

LEMMA 3. Constructing the lookup table takes $O(cn^2)$ time.

PROOF. The lookup table can be constructed using dynamic programming. Given F_m, F_M , and c , we need to consider whether F_m is chosen or not. Hence the construction is based on the following recurrence relation $wt(F_m, F_M, c)$ defined as

$$\begin{aligned} & 0 && \text{if } c = 0 \\ & +\infty && \text{if } F_m > F_M \\ & +\infty && \text{if } F_m > n - c + 1 \\ & \min(wt(F_m + 1, F_M, c), && \text{otherwise} \\ & \quad w(F_m) + wt(F_m + 1, n - c + 2, c - 1)) \end{aligned}$$

The first three cases are the base cases. It takes constant time to compute every entry of the table and hence the construction time is the same as the table size which is $O(cn^2)$. \square

3.2 Finding the Successor

Following [7], we specify the successor algorithm.

SPECIFICATION 3 (SUCC). For a unary constraint \mathcal{C} and a length-lex interval X satisfying $hs(\mathcal{C})(X)$, function $succ(\mathcal{C})(X)$ returns the smallest set $x \in X$ in the length-lex ordering satisfying the constraint, i.e., $succ(\mathcal{C})(X) \equiv \min_{\preceq} x \in X : \mathcal{C}(x)$.

Algorithm 2 $succ(kp(w, b))(X_{pf} = pf\langle P, F, V, c \rangle)$

1: $s_{1..|P|} \leftarrow P$
2: $l, h \leftarrow F_m, F_M$
3: $b' \leftarrow b - w(P)$
4: **for** $i = |P| + 1$ **to** c **do**
5: $s_i \leftarrow \min_{f \in l..h} hs(kp(w, b'))(pf(\emptyset, \{f\}, \{f, \dots, n\}, c - i + 1))$
6: $l, h \leftarrow s_i + 1, n$
7: $b' \leftarrow b' - w(s_i)$
8: **return** s

Algorithm 3 $succ(kp(w, b))(X = \langle m, M \rangle)$

1: $[X_{pf}^1, \dots, X_{pf}^i] \leftarrow decomp(m, M, \emptyset, n)$
2: **for** $X_{pf} = X_{pf}^1$ **to** X_{pf}^i **do**
3: **if** $hs(kp(w, b))(X_{pf})$ **then**
4: **return** $succ(kp(w, b))(X_{pf})$
5: **return** \perp

Algorithm 2 implements Specification 3 where the constraint \mathcal{C} is $kp(w, b)$, i.e., it constructs the (length-lex) smallest feasible set in the given PF-closed interval. It first assigns the prefix set to s (line 1) and then iterates (lines 4–7) until the set is complete. Line 5 is essentially a search for the value of the i -th element that satisfy the feasibility routine. It can be done by a binary search on the range $[l, h]$ with at most $O(\log(h - l + 1))$ table lookups. In the first iteration, the value is chosen from F (line 2) and, in the remaining ones, the value is chosen from V (line 6), making sure that the elements are increasing. Note that b' denotes the remaining capacity at all points of the algorithm.

EXAMPLE 5 (SUCC). We use the settings from Example 3 for which there is a solution in the interval. After the initialization, $s = \{1\}$, $l, h = 4, 6$. l and h is the F -set restriction, the remaining capacity is $b' = 5$. The algorithm constructs the set by iterating from the second element to the last. Item 4 is good since $wt(4, 4, 3) = 3 \leq 5$, which means it is possible to construct a 3-set when we include item 4 as the second element. It then assigns 4 to s_2 , sets the new range, and update the capacity by subtracting the weight of the current choice. In the third iteration, the remaining capacity is 4. Item 5 is not possible since $w(5) = 5$ and exceeds the capacity. Item 6 is good since $wt(6, 6, 2) = 2 < 4$ and the algorithm assigns 6 to the third element. The algorithm eventually returns the set $\{1, 4, 6, 7\}$ with weight 6, which is the smallest in the length-lex ordering in the interval that satisfies the knapsack constraint. Notice it is not necessarily the set with minimum weight: The set found in Example 3 is $\{1, 4, 6, 8\}$ with weight 5.

LEMMA 4. $succ(kp(w, b))(X_{pf})$ takes $O(c \log n)$.

PROOF. Lines 1-3 takes $O(c)$. In Line 5, we search for the minimum f by a binary search over the range $[l, h]$, the largest range is smaller than n . It invokes at most $O(\log n)$ calls to the feasibility routine which only takes $O(1)$ time since the prefix is empty. Since there are at most $O(c)$ iterations, the total time is $O(c \log n)$. \square

3.3 The Bound-Consistency Algorithm

Algorithm 3 takes an length-lex interval and returns the smallest set satisfying the knapsack constraint. It first decomposes the length-lex interval into a list of PF-closed interval, then finds the

¹The value is $+\infty$ when no c -set can be constructed according to the given criteria

smallest PF-closed interval that contains a feasible set, and constructs the smallest feasible set using Algorithm 2.

LEMMA 5. $\text{succ}\langle kp(w, b)\rangle(X)$ takes $O(c^2 + c \log n)$.

PROOF. Algorithm *decomp* in line 1 takes $O(c^2)$ and returns a list of size $O(c)$. The feasibility routine on line 3 takes $O(c)$ and line 4 takes $O(c \log n)$. Line 3 runs at most $O(c)$ times and line 4 runs at most once. Hence the total time is $O(c^2 + c \log n)$. \square

4. AMORTIZATION

This section shows how to reduce the time for enforcing bound consistency to $O(c \log n)$ by amortizing the cost of the decomposition. The key idea is to observe that length-lex intervals whose upper bounds are maximum have some nice structure in their decomposition. Therefore, instead of decomposing a length-lex interval $\langle m, M \rangle$, the algorithm “decomposes” $\langle m, \nabla_c \rangle$ where ∇_c is the maximum c -set in the universe (i.e. $\{n - c + 1, \dots, n\}$). More precisely, once intervals of the form $\langle m, \nabla_c \rangle$ are considered, we can apply the two-phase algorithmic structure from [4] to locate first the PF-closed interval that contains the smallest set before constructing the actual set. Because the algorithm is not applied to $\langle m, \nabla_c \rangle$, it may happen that the result set m' is greater than M in which the domain is inconsistent and the algorithm returns \perp .

4.1 The Decomposition Revisited

The decomposition algorithm [7] partitions an interval X into a head H , a body B , and a tail T . However, when the upper bound is maximal, the resulting PF-closed intervals exhibit a nice structure.

EXAMPLE 6. Given $U = \{1, \dots, 8\}$, the length-lex interval $\langle \{1, 3, 5, 6\}, \nabla_4 = \{5, 6, 7, 8\} \rangle$ is decomposed into 5 PF-closed intervals.

$$\begin{array}{l} pf\langle \{1, 3, 5\}, \{6\}, \{6, \dots, 8\}, 4 \rangle \\ pf\langle \{1, 3, 5\}, \{7, 8\}, \{7, 8\}, 4 \rangle \\ pf\langle \{1, 3\}, \{6, 7\}, \{6, \dots, 8\}, 4 \rangle \\ pf\langle \{1\}, \{4, \dots, 6\}, \{4, \dots, 8\}, 4 \rangle \\ pf\langle \{\}, \{2, \dots, 5\}, \{2, \dots, 8\}, 4 \rangle \end{array}$$

Formally, a length-lex interval $\langle \{m_1, m_2, \dots, m_c\}, \nabla_c \rangle$ can be decomposed into at most $c + 1$ PF-closed intervals which falls into two categories: First the lower bound m itself

$$pf\langle m_{1..c-1}, \{m_c\}, \{m_c, \dots, n\}, c \rangle$$

and then remaining PF-closed intervals

$$pf\langle m_{1..i-1}, \{m_i + 1, \dots, n - c + i\}, \{m_i + 1, \dots, n\}, c \rangle$$

if $m_i + 1 \leq n - c + i$ for $i \in \{1, \dots, c\}$.

Observe that the prefixes in the second category decreases by one element each time, which will allow the algorithm to avoid having to compute $w(P)$ from scratch, amortizing its cost across the decomposition.

4.2 The Location Phase

The location routine combines the decomposition and feasibility routines and does not explicitly construct a list of PF-closed intervals. Its goal is similar to line 3 of Algorithm in that it finds the first PF-closed interval containing a feasible set. More precisely, it returns 0 if the lower bound is a solution, -1 if there are no solution in $\langle m, \nabla_c \rangle$, and i if the smallest PF-closed interval has $|P| = i - 1$.

SPECIFICATION 4 (*locate_{succ}*). Given $X = \langle m, \nabla_c \rangle$, Algorithm *locate_{succ}*(\mathcal{C})(X) returns 0 if $\mathcal{C}(m)$, \perp if $\nexists s \in X : \mathcal{C}(s)$, and

$$\max_{i \in 1..c} hs(\mathcal{C})(pf\langle m_{1..i-1}, \{m_i + 1, n - c + i\}, \{m_i + 1, \dots, n\}, c \rangle)$$

otherwise.

Algorithm 4 *locate_{succ}*($kp(w, b)\rangle(X = \langle m, M \rangle)$)

```

1:  $c = \lfloor m \rfloor$ 
2:  $W \leftarrow w(m_{1..c})$ 
3: if  $W \leq b$  then
4:   return 0
5: for  $i = c$  to 1 do
6:    $W \leftarrow W - w(m_i)$ 
7:   if  $wt(m_i + 1, n - c + 1, c) \leq b - W$  then
8:     return  $i$ 
9: return  $\perp$ 

```

Algorithm 4 implements Specification 4 for the knapsack constraint. The bottleneck in the earlier algorithm is the time it takes to calculate the prefix weight for each of the PF-closed intervals. We solve the issue by incrementally updates the weight, since we have shown that the PF-closed intervals are closely related. Line 3 determines if the current lower bound m is feasible. If not, the algorithm traverses the list of PF-closed intervals in the second category by decreasing sizes of the prefixes. Variable W maintains the prefix weight which decreases in each iteration (Line 6). As every pair of PF-closed intervals differs by at most one element, updating W takes constant time. Line 7 applies the feasibility routine to determine if the corresponding PF-closed interval contains a solution. If none of the above test returns positive result, there is no solution and the routine returns \perp .

EXAMPLE 7 (LOCATE). Example 6 demonstrated the decomposition of interval $\langle \{1, 3, 5, 6\}, \{4, 6, 7, 8\} \rangle$. Since the lower bound is infeasible, W is initially to its weight (11). The for-loop iterates from c down to 1. The first iteration ($i = c$) corresponds to the PF-closed interval with prefix length $c - 1$, the difference between this prefix and the lower bound is $w(m_c)$, which is deducted it from W , which remains 11 in this case. This PF-closed interval is also infeasible. In the next iteration, 5 is removed from the prefix and W becomes 6. This feasibility test fails again and we move to the next PF-closed interval. Here, $w(3) = 4$ is subtracted from W which becomes 1 and $wt(4, 6, 3) = 3$. Hence there exists a solution in this PF-closed interval and the locate routine returns 2.

LEMMA 6. Algorithm 4 runs in $O(c)$ time.

PROOF. Line 2 takes $O(c)$, since computing $w(m_{1..c-1})$ takes $O(c)$. Lines 5–8 takes $O(c)$ time, since every line takes constant time. Hence, the total time is $O(c)$. \square

4.3 Finding the Successor

Finding the successor consists essentially in applying the location routine and in constructing a minimum feasible set using Algorithm 3. Since we transformed $\langle m, M \rangle$ into $\langle m, \nabla_c \rangle$, the newly constructed set may exceed the upper bound of the original interval. In such a case, the algorithm returns a \perp that indicates inconsistency. The specification in [7] invokes the *succ* algorithm assuming that there exists a solution. Since, in this amortized version, the precondition does not necessarily hold, it is necessary to change the specification.

SPECIFICATION 5 (*succ_{Amortized}*).

$$\begin{aligned} succ_A\langle \mathcal{C} \rangle(X = \langle m, M \rangle) \\ \equiv \begin{cases} \perp & \text{if } \nexists x \in X \text{ s.t. } \mathcal{C}(x) \\ \min_{\leq} \{x \in X \mid \mathcal{C}(x)\} & \end{cases} \end{aligned}$$

Algorithm 5 *succAmortized*($\langle kp(w, b) \rangle$)($X = \langle m, M \rangle$)

```
1:  $c = |m|$ 
2:  $i \leftarrow locate(\langle kp(w, b) \rangle)(\langle m, \nabla_c \rangle)$ 
3: if  $i = -1$  then
4:   return  $\perp$ 
5: else if  $i = 0$  then
6:   return  $m$ 
7: else
8:    $m' \leftarrow succ(\langle kp(w, b) \rangle)(pf(\langle m_{1,\dots,i-1}, \{m_{i+1}+1, \dots, n-c+i\}, \{m_{i+1}+1, \dots, n\}, c))$ 
9:   if  $m' \preceq M$  then
10:    return  $m'$ 
11:  else
12:    return  $\perp$ 
```

Algorithm 5 implements Specification 5 for knapsack constraints along the scheme sketched above.

EXAMPLE 8. Consider $X = \langle \{1, 3, 5, 6\}, \{1, 3, 7, 8\} \rangle$ and a universe $U = \{1, \dots, 8\}$. The location routine transforms the length-lex interval into $\langle \{1, 3, 5, 6\}, \{5, 6, 7, 8\} \rangle$, and the successor algorithm returns the minimum feasible set $\{1, 4, 6, 7\}$. However, this set is greater than $\{1, 3, 7, 8\}$ in length-lex ordering, indicating domain inconsistency.

THEOREM 1. Enforcing bound consistency on unary knapsack constraint on length-lex set variable takes $O(c \log n)$ time.

PROOF. Line 2 takes $O(c)$. Line 8 takes $O(c \log n)$. Line 9 takes $O(c)$. Hence $O(c \log n)$ in total. \square

5. DISCUSSION AND RELATED WORK

5.1 Special Cases

First observe that our algorithm directly applies to constraints of the form

$$\sum_{e \in X} w(e) \leq b.$$

since our algorithm makes no assumptions on the weights or on b . Moreover, we can easily implement other constraints by manipulating the weight function. We demonstrate the algorithm for finding the first successor that includes all elements of R (Algorithm 3 in [4]) and the first successor that excludes all elements of E (Algorithm 5 in [4]).

Suppose $in(S, R) \equiv \{s \in S \mid R \subseteq s\}$ where S is a set variable.

THEOREM 2. $in(R)$ is equivalent to $kp(w, b)$ where $b = -|R|$, $w(r) = -1$ if $r \in R$ and $w(r) = 0$ if $r \notin R$.

PROOF. There are exactly $|R|$ elements (with weight -1) in set R , $w(s) = -|R|$ if and only if $R \subseteq s$ \square

Suppose $ex(S, E) \equiv \{s \in S \mid s \cap E = \emptyset\}$ where S is a set variable.

THEOREM 3. $ex(E)$ is equivalent to $kp(w, b)$ where $b = 0$, $w(e) = 1$ if $e \in E$ and $w(e) = 0$ if $e \notin E$.

PROOF. If a set contains an element in the excluded list E , its weight is greater than zero and thus exceeds the bound $b = 0$. \square

The complexity of our algorithm actually improves the result from [4], when $|E|$ is larger than c .

5.2 Related Work

Reference [2] proposed a polynomial algorithm for enforcing approximated bound consistency for two knapsack constraints over the same length-lex set variable. In particular, it showed that approximated ϵ -length-lex* bounds consistency for two knapsack constraints can be achieved in time $O(\frac{n^4}{\epsilon})$. The authors mentioned that, although it is polynomial, their algorithm is impractical.

This paper provides an alternative to their impractical algorithm, by posting the two constraints independently. Indeed, since the algorithm enforces bound consistency on both (unary) constraints, the fixpoint is guaranteed to ensure that the two bounds will be solutions of the conjunction. Since our algorithms are extremely fast once posted, entail almost no cost when no pruning occurs, and will naturally benefit from the filtering performed by other problem constraints, this alternative is particularly appealing. Moreover, slow convergence (if any) can always be detected if necessary in order to bound the number of iterations of the fixpoint algorithms.

It should also be pointed out that the algorithm in [2] does not allow zero weight items, mostly because of the fixed-parameter approximation scheme depends on the weight of item to scale down the overall weight. Although this is not a problem for pure knapsack problems, this becomes a serious issue when cardinality and lexicographic constraints are taken into account, since the inclusion/exclusion of a zero-weight item make a lot of sense now and cannot be ignored by the knapsack constraint.

6. CONCLUSION

This paper studied how to enforce bound consistency on a knapsack constraint over a length-lex set variable. Knapsack constraints are a building block of many applications in which set variables occur including configurations and network design problems. The paper showed how to enforce bound consistency on knapsack constraints in $O(c \log n)$ time with a one-time $O(cn^2)$ preprocessing time when the constraint is posted. Because of the practicability of the algorithm, the results also provide the basis for an appealing alternative to the $O(\frac{n^4}{\epsilon})$ algorithm of [2] for enforcing approximated bound consistency on the conjunction of two knapsack constraints over the same length-lex set variable.

7. REFERENCES

- [1] Azevedo, F., Barahona, P. 2000. Modelling Digital Circuits Problems with Set Constraints. in *CL-2000*.
- [2] Malitsky, Y., Sellmann, M., Van Hoeve, W. 2008. Length-Lex Bounds Consistency for Knapsack Constraints. In *Proc. CP'08*.
- [3] Gervet, C. 1997. Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. In *Constraints journal*, volume 1(3). Kluwer.
- [4] Gervet, C., Van Hentenryck, P. 2006. Length-Lex Ordering for Set CSPs. In *Proc. AAAI'06*.
- [5] Puget, J-F. 1992. PECOS a High Level Constraint Programming Language In *Proc. of Spicis*.
- [6] Sadler, A., Gervet, C. 2007. Enhancing Set Constraint Solvers with Lexicographic Bounds. In *Journal of Heuristics*, 4(1), 2007.
- [7] Van Hentenryck, P., Yip, J., Gervet, C. and Dooms, G., 2008. Bound Consistency for Binary Length-Lex Set Constraints. In *Proc. AAAI'08*.