Three Generalizations of the FOCUS Constraint
Nina Narodytska, Thierry Petit, Mohamed Siala, Toby Walsh

To cite this version:

HAL Id: hal-00871750
https://hal.archives-ouvertes.fr/hal-00871750
Submitted on 15 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Three Generalizations of the FOCUS Constraint

Nina Narodytska  
NICTA and UNSW  
Sydney, Australia  
ninan@cse.unsw.edu.au

Thierry Petit  
LINA-CNRS  
Mines-Nantes, INRIA  
nantes, France  
thierry.petit@mines-nantes.fr

Mohamed Siala  
LAAS-CNRS  
Univ de Toulouse, INSA  
Toulouse, France  
msiala@laas.fr

Toby Walsh  
NICTA and UNSW  
Sydney, Australia  
toby.walsh@nicta.com.au

Abstract

The FOCUS constraint expresses the notion that solutions are concentrated. In practice, this constraint suffers from the rigidity of its semantics. To tackle this issue, we propose three generalizations of the FOCUS constraint. We provide for each one a complete filtering algorithm as well as discussing decompositions.

1 Introduction

Many discrete optimization problems have constraints on the objective function. Being able to represent such constraints is fundamental to deal with many real world industrial problems. Constraint programming is a promising approach to express and filter such constraints. In particular, several constraints have been proposed for obtaining well-balanced solutions [Pesant and Régis, 2005; Schaus et al., 2007; Petit and Régis, 2011]. Recently, the FOCUS constraint [Petit, 2012] was introduced to express the opposite notion. It captures the concept of concentrating the high values in a sequence of variables to a small number of intervals. We recall its definition. Throughout this paper, $X = \{x_0, x_1, \ldots, x_{n-1}\}$ is a sequence of variables and $s_{i,j}$ is a sequence of indices of consecutive variables in $X$, such that $s_{i,j} = [i, i+1, \ldots, j]$, $0 \leq i \leq j < n$. We let $|E|$ be the size of a collection $E$.

Definition 1 (Petit, 2012). Let $y_c$ be a variable. Let $k$ and $\text{len}$ be two integers, $1 \leq \text{len} \leq |X|$. An instantiation of $X \cup \{y_c\}$ satisfies FOCUS($X, y_c, \text{len}, k$) if there exists a set $S_X$ of disjoint sequences of indices $s_{i,j}$ such that three conditions are all satisfied: (1) $|S_X| \leq y_c$  (2) $\forall x_i \in X, x_i > k \Leftrightarrow \exists s_{i,j} \in S_X$ such that $l \in s_{i,j}$  (3) $\forall s_{i,j} \in S_X, j - i + 1 \leq \text{len}$

FOCUS can be used in various contexts including cumulative scheduling problems where some excesses of capacity can be tolerated to obtain a solution [Petit, 2012]. In a cumulative scheduling problem, we are scheduling activities, and each activity consumes a certain amount of some resource. The total quantity of the resource available is limited by a capacity. Excesses can be represented by variables [De Clercq et al., 2011]. In practice, excesses might be tolerated by, for example, renting a new machine to produce more resource. Suppose the rental price decreases proportionally to its duration: it is cheaper to rent a machine during a single interval than to make several rentals. On the other hand, rental intervals have generally a maximum possible duration. FOCUS can be set to concentrate (non null) excesses in a small number of intervals, each of length at most $\text{len}$.

Unfortunately, the usefulness of FOCUS is hindered by the rigidity of its semantics. For example, we might be able to rent a machine from Monday to Sunday but not use it on Friday. It is a pity to miss such a solution with a smaller number of rental intervals because FOCUS imposes that all the variables within each rental interval take a high value. Moreover, a solution with one rental interval of two days is better than a solution with a rental interval of four days. Unfortunately, FOCUS only considers the number of disjoint sequences, and does not consider their length.

We tackle those issues here by means of three generalizations of FOCUS. SPRINGYFOCUS tolerates within each sequence in $s_{i,j} \in S_X$ some values $v \leq k$. To keep the semantics of grouping high values, their number is limited in each $s_{i,j}$ by an integer argument. WEIGHTEDFOCUS adds a variable to count the length of sequences, equal to the number of variables taking a value $v > k$. The most generic one, WEIGHTEDSPRINGYFOCUS, combines the semantics of SPRINGYFOCUS and WEIGHTEDFOCUS. Propagation of constraints like these complementary to an objective function is well-known to be important [Petit and Poder, 2008; Schaus et al., 2009]. We present and experiment with filtering algorithms and decompositions therefore for each constraint.

2 Springy FOCUS

In Definition 1, each sequence in $S_X$ contains exclusively values $v > k$. In many practical cases, this property is too strong. Consider one simple instance of the problem in the introduction, in Figure 1, where one variable $x_i \in X$ is defined per point in time (e.g., one day), to represent excesses of capacity. Initially, 4 activities are fixed and one activity $a$ remains to be scheduled (drawing A), of duration 5 and that can start from day 1 to day 5. If FOCUS($X, y_c = 1, 5, 0$) is imposed then $a$ must start at day 1 (solution B). We have one 5 day rental interval. Assume now that the new machine may not be used every day. Solution (C) gives one rental of 3 days instead of 5. Furthermore, if $\text{len} = 4$ the problem will have no solution using FOCUS, while this latter solution still exists in practice. This is paradoxical, as relaxing the condition...
that sequences in the set $S_X$ of Definition 1 take only values $v > k$ deteriorates the concentration power of the constraint. Therefore, we propose a soft relaxation of FOCUS, where at most $h$ values less than $k$ are tolerated within each sequence in $S_X$.

**Definition 2.** Let $y_c$ be a variable and $k$, len be three integers, $1 \leq len \leq |X|$, $0 \leq h < len−1$. An instantiation of $X \cup \{y_c\}$ satisfies SPRINGFOCUS($X, y_c, len, k, h$) if there exists a set $S_X$ of disjoint sequences of indices $s_{i,j}$ such that four conditions are all satisfied: (i) $|S_X| \leq y_c$, (ii) $\forall x_i \in X$, $x_i > k \Rightarrow \exists s_{l,i} \in S_X$ such that $l \in s_{i,j} \in S_X$. $j < i \leq len$, $x_i > k$ and $x_j > k$. (iii) $\forall s_{i,j} \in S_X$, \{i ∈ $s_{i,j}, x_i < k$\} ≤ h

Bounded consistency (BC) on SPRINGFOCUS is equivalent to domain consistency: any solution can be turned into a solution that only uses the lower bound $min(x_i)$ or the upper bound $max(x_i)$ of the domain $D(x_i)$ of each $x_i \in X$ (this observation was made for FOCUS [Petit, 2012]). Thus, we propose a BC algorithm. The first step is to traverse X from $x_0$ to $x_{n-1}$, to compute the minimum possible number of disjoint sequences in $S_X$ (a lower bound for $y_c$), the focus cardinality, denoted $fc(X)$. We use the same notation for subsequences of $X, fc(X)$ depends on $k, len$ and $h$.

**Definition 3.** Given $x_i \in X$, we consider three quantities.

1. $p(x_i, v_c) = \min$ is the focus cardinality of $[x_0, x_1, \ldots, x_i, v_i]$ assuming $x_i ≤ k$, and $|S_{x_0, x_1, \ldots, x_i}| \neq l$. \( p(x_i, v_c) \) is the focus cardinality of $[x_0, x_1, \ldots, x_i]$, assuming $x_i ≤ k$ and $\exists s_{i,j} \in S_{x_0, x_1, \ldots, x_i}$.

Any quantity is equal to $n + 1$ if the domain $D(x_i)$ of $x_i$ makes not possible the considered assumption.

**Property 1.** $p(x_0, v_c) = p(x_{n-1}, v_c) = n + 1$, and $fc(X) = \min(p(x_{n-1}, v_c), p(x_{n-1}, v_c))$.

**Proof.** By construction from Definitions 2 and 3.

To compute the quantities of Definition 3 for $x_i \in X$, we use $plen(x_i)$, the minimum length of a sequence in $S_{x_0, x_1, \ldots, x_i}$ containing $x_i$ among instantiations of $[x_0, x_1, \ldots, x_i]$ where the number of sequences is $fc([x_0, x_1, \ldots, x_i])$, $plen(x_i) = 0$ if $\forall s_{i,j} \in S_{x_0, x_1, \ldots, x_i}, j \neq l, card(x_i)$ is the minimum number of values $v ≤ k$ in the current sequence in $S_{x_0, x_1, \ldots, x_i}$, equal to 0 if $\forall s_{i,j} \in S_{x_0, x_1, \ldots, x_i}, j \neq l, card(x_i)$ assumes that $x_i > k$. It has to be decreased by one if $x_i ≤ k$. For sake of space, the proofs of the next four lemmas are given in a technical report [Naroditska et al., 2013].

**Algorithm 1: MINCARDS($X, len, k, h$): Integer matrix**

1. pre := new Integer[|X||len|];
2. for $l \in 0..n-1$ do
3. \( pre[0,l] := new Integer[2]; \)
4. for $j \in 1..3$ do $pre[1,j] := new Integer[1];$
5. Initialization Lemma 1
6. for $l \in 1..n-1$ do Lemmas 2, 3, and Propositions 1 and 2.
7. return pre;

**Lemma 1** (initialization). $p(x_0, v_c) = 0$ if $min(x_0) ≤ k$, $p(x_0, v_c) = n + 1$ otherwise; $p_s(x_0, v_c) = n + 1$; $p(x_0, v_c) = 1$ if $max(x_0) > k$ and $n + 1$ otherwise; $plen(x_0) = 1$ if $max(x_0) > k$ and 0 otherwise; $card(x_0) = 0$.

**Lemma 2** ($p(x_i, v_c)$). If $min(x_i) ≤ k$ then $p(x_i, v_c) = min(p(x_{i-1}, v_c), p(x_{i-1}, v_c))$, else $p(x_i, v_c) = n + 1$.

**Lemma 3** ($p_s(x_1, v_c)$). If $min(x_i) > k, p_s(x_1, v_c) = n + 1$. Otherwise, if $plen(x_i-1) \in \{0, len-1, len\}$ $\land card(x_i-1) = 1$ then $p_s(x_i, v_c) = n + 1$, else $p_s(x_i, v_c) = min(p_s(x_{i-1}, v_c), p(x_{i-1}, v_c), p(x_{i-1}, v_c) + 1)$.

**Proposition 1** ($plen(x_i)$, (by construction)) If $min(p(x_{i-1}, v_c), p(x_{i-1}, v_c)) ≤ p(x_{i-1}, v_c) + 1 \land plen(x_i-1) < len$ then $plen(x_i) = plen(x_i-1) + 1$. Otherwise, if $p(x_i, v_c) < n + 1$ then $plen(x_i) = 1$, else $plen(x_i) = 0$.

**Proposition 2** ($card(x_i)$). If $plen(x_i) = 1$ then $card(x_i) = 0$. Otherwise, if $p(x_i, v_c) = n + 1$ then $card(x_i) = card(x_i-1)+1$, else $card(x_i) = card(x_i-1)$.

Algorithm 1 implements the lemmas with $pre[l][0][0] = p(x_i, v_c), pre[l][0][1] = p_s(x_1, v_c), pre[l][1] = p(x_i, v_c), pre[l][2] = plen(x_i), pre[l][3] = card(x_i)$. The principle of Algorithm 2 is the following. First, $lb = fc(X)$ is computed with $x_0$. We execute Algorithm 2 from $x_0$ to $x_{n-1}$ and conversely (arrays $pre$ and $suf$). We thus have for each quantity two values for each variable $x_i$. To aggregate them, we implement regret mechanisms directly derived from Propositions 1 and 3 according to the parameters $len$ and $h$.

Line 4 is optional but it avoids some work when the variable $y_c$ is fixed, thanks to the same property as FOCUS (see Petit, 2012). Algorithm 2 performs a constant number of traversals of the set $X$. Its time complexity is $O(n)$, which is optimal.

**3 Weighted FOCUS**

We present WEIGHTEDFOCUS, that extends FOCUS with a variable $z_c$, limiting the sum of lengths of all the sequences in $S_X$, i.e., the number of variables covered by a sequence in $S_X$. It distinguishes between solutions that are equivalent with respect to the number of sequences in $S_X$ but not with respect to their length, as Figure 2 shows.

**Definition 4.** Let $y_c$ and $z_c$ be two integer variables and $k$, len be two integers, such that $1 \leq len \leq |X|$.
Algorithm 2: FILTERING($X, y_c, \text{len}, k, h$): Set of variables

1. $pre := \text{MINCARDS}(X, \text{len}, k, h)$;
2. $\text{integer}\ b := \text{min}\{\text{pre}[n-1][0][0], \text{pre}[n-1][1]\}$;
3. if $\text{min}(y_c) < b \text{ then } D(y_c) := D(y_c) \setminus \{\text{min}(y_c), b\}$;
4. if $\text{min}(y_c) = \text{max}(y_c)$ then
5. $\text{suf} := \text{MINCARDS}(\{x_0, x_1, \ldots, x_{n-1}\}, \text{len}, k, h)$;
6. for $l \in 0, n-1$ do
7. if $\text{pre}[l][0][0] + \text{max}(y_c) > \text{max}(y_c)$ then
8. Integer regret := 0; Integer add := 0;
9. if $\text{pre}[l][1] \text{ or } \text{pre}[l][2] \text{ then } \text{add} := \text{add} + 1$;
10. if $\text{add} = 0 \text{ and } \text{regret} = 0 \text{ then } D(x_l) := D(x_l) \setminus \{\text{min}(x_l), k\}$;
11. if $\text{add} = 1 \text{ and } \text{regret} = 1$ then
12. if $\text{max}(y_c) > \text{max}(y_c)$ then
13. $D(x_l) := D(x_l) \setminus \{\text{max}(x_l), k\};$
14. if $\text{add} = 1 \text{ and } \text{regret} = 0$ then
15. if $\text{add} = 1 \text{ and } \text{regret} = 1$ then
16. $D(x_l) := D(x_l) \setminus \{\text{max}(x_l), k\};$
17. return $X \cup \{y_c\};$

$X$. An instantiation of $X \cup \{y_c\} \cup \{z_c\}$ satisfies $\text{WEIGHTEDFOCUS}(X, y_c, \text{len}, k, z_c)$ iff there exists a set $S_X$ of disjoint sequences of indices $s_{i,j}$ such that four conditions are all satisfied: (1) $|S_X| \leq y_c$ (2) $\forall x_i \in X, x_i > k \iff \exists s_{i,j} \in S_X \text{ such that } i \in s_{i,j}$ (3) $\forall s_{i,j} \in S_X, j = i + 1 \leq \text{len}$ (4) $\sum_{s_{i,j} \in S_X} |s_{i,j}| \leq z_c$.

Definition 5 (Pieti, 2012). Given an integer $k$, a variable $x_i \in X$ is: Penalizing ($P_k$), iff $\text{min}(x_i) > k$. Neutral, ($N_k$), iff $\text{max}(x_i) \leq k$. Undetermined, ($U_k$), otherwise. We say $x_i \in P_k$ if $x_i$ is labeled $P_k$, and similarly for $U_k$ and $N_k$.

Dynamic Programming (DP) Principle Given a partial instantiation $I_X$ of $X$ and a set of sequences $S_X$ that covers all penalizing variables in $I_X$, we consider two terms: the number of variables in $P_k$ and the number of undetermined variables, in $U_k$, covered by $S_X$. We want to find a set $S_X$ that minimizes the second term. Given a sequence of variables $s_{i,j}$, the cost $\text{cst}(s_{i,j})$ is defined as $\text{cst}(s_{i,j}) = \{p|x_p \in U_k, x_p \in s_{i,j}\}$. We denote cost of $S_X$, $\text{cst}(S_X)$, the sum $\text{cst}(S_X) = \sum_{s_{i,j} \in S_X} \text{cst}(s_{i,j})$. Given $I_X$ we consider $|P_k| = |\{x_i \in P_k\}|$. We have: $\sum_{s_{i,j} \in S_X} |s_{i,j}| = \sum_{s_{i,j} \in S_X} \text{cst}(s_{i,j}) + |P_k|$. We start with explaining the main difficulty in building a propagator for $\text{WEIGHTEDFOCUS}$. The constraint has two optimization variables in its scope and we might not have a solution that optimizes both variables simultaneously.

Example 1. Consider the set $X = \{x_0, x_1, \ldots, x_5\}$ with domains $\{1, 0, 1\} \cup \{0, 1, 1\}$ and $\text{WEIGHTEDFOCUS}(X, \{2, 3\}, 3, 0, \{0, 6\})$. Solution $S_X = \{s_{0,2}, s_{3,5}\}$, $z_c = 6$, minimizes $y_c = 2$, while solution $S_X = \{s_{0,1}, s_{2,3}, s_{3,5}\}$, $y_c = 3$, minimizes $z_c = 4$.

Example 1 suggests that we need to fix one of the two optimization variables and only optimize the other one. Our algorithm is based on a dynamic program [Dasgupta et al., 2006]. For each prefix of variables $x_0, x_1, \ldots, x_j$ and given a cost value $c$, it computes a cover of focus cardinality, denoted $S_{c,j}$, which covers all penalized variables in $x_0, x_1, \ldots, x_j$ and has cost exactly $c$. If $S_{c,j}$ does not exist we assume that $S_{c,j} = \infty$. $S_{c,j}$ is not unique as Example 2 demonstrates.

Example 2. Consider $X = \{x_0, x_1, \ldots, x_7\}$ and $\text{WEIGHTEDFOCUS}(X, \{2, 3\}, 5, 0, \{7\})$, with $D(x_i) = \{1\}$, $i \in I, I = \{0, 2, 3, 5, 7\}$ and $D(x_i) = \{0, 1\}, i \in \{0, 1, \ldots, 7\} \setminus I$. Consider the subsequence of variables $x_0, \ldots, x_5$ and $S_{1,5}$. There are several sets of minimum cardinality that cover all penalized variables in the prefix $x_0, \ldots, x_5$ and has cost 2, e.g. $S_{1,5} = \{s_{0,2}, s_{3,5}\}$ or $S_{1,5} = \{s_{0,4}, s_{5,5}\}$. Assume we sort sequences by their starting points in each set. We note that the second set is better if we want to extend the last sequence in this set as the length of the last sequence $s_{5,5}$ is shorter compared to the length of the last sequence in $S_{1,5}$, which is $s_{3,5}$.

Example 2 suggests that we need to put additional conditions on $S_{c,j}$ to take into account that some sets are better than others. We can safely assume that none of the sequences in $S_{c,j}$ starts at undetermined variables as we can always set it to zero. Hence, we introduce a notion of an ordering between sets $S_{c,j}$ and define conditions that this set has to satisfy.

Ordering of sequences in $S_{c,j}$. We introduce an order over sequences in $S_{c,j}$. Given a set of sequences in $S_{c,j}$ we sort them by their starting points. We denote $\text{last}(S_{c,j})$ the last sequence in $S_{c,j}$ in this order. If $x_i \in \text{last}(S_{c,j})$ then $\text{last}(S_{c,j}) = \text{last}(S_{c,j})$, otherwise $\text{last}(S_{c,j}) = \infty$.

Ordering of sets $S_{c,j}, c \in [0, \text{max}(z_c)], j \in \{0, 1, \ldots, n-1\}$. We define a comparison operation between two sets $S_{c,j}$ and $S_{c,j'}$, $S_{c,j} \leq S_{c,j'}$ iff $|S_{c,j}| < |S_{c,j'}|$ or $|S_{c,j}| = |S_{c,j'}|$ and $\text{last}(S_{c,j}) \leq \text{last}(S_{c,j})$. Note that we do not take account of cost in the comparison as the current definition is sufficient for us. Using this operation, we can compare all sets $S_{c,j}$ and $S_{c,j'}$ of the same cost for a prefix $x_0, \ldots, x_j$. We say that $S_{c,j}$ is optimal iff satisfies the following 4 conditions.

Proposition 3 (Conditions on $S_{c,j}$).

1. $S_{c,j}$ covers all $P_k$ variables in $x_0, x_1, \ldots, x_j$.
2. $\text{cst}(S_{c,j}) = c$.
3. $\forall s_{i,j} \in S_{c,j}, x_h \notin U_k$.
4. $S_{c,j}$ is the first set in the order among all sets that satisfy conditions 1, 2.
Figure 3: Representation of one step of Algorithm 3

not exist. However, if \(|S_{c,j}| = |S_{c',j'}|, c = c' \) and \(j = j'\), then \(last(S_{c,j}) = last(S_{c',j'})\).

Example 3. Consider WEIGHTEDFOCUS from Example 2
Consider the subsequence \([x_0, x_1]\). \(S_{0,1} = \{S_{0,0}\}\), \(S_{1,1} = \{S_{0,1}\}\). \(S_{2,1}\) does not exist. Consider the subsequence \([x_0, \ldots, x_5]\). We have \(S_{0,5} = \{S_{0,0}, S_{0,2}, S_{0,3}, S_{0,5}\}\), \(S_{1,5} = \{S_{0,4}, S_{0,5}\}\) and \(S_{2,5} = \{S_{0,3}, S_{0,5}\}\). By definition, \(last(S_{0,5}) = S_{0,5}\), \(last(S_{1,5}) = S_{0,5}\) and \(last(S_{2,5}) = S_{0,5}\). Consider the set \(S_{1,5}\). Note that there exists another set \(S_{1,5}' = \{S_{0,0}, S_{0,2}, S_{0,3}, S_{0,5}\}\) that satisfies conditions \([19]\). Hence, it has the same cardinality as \(S_{1,5}\) and the same cost. However, \(S_{1,5} < S_{1,5}'\) as \(|last(S_{1,5})| = 1 < |last(S_{1,5}')| = 3\).

Bounds disentailment Each cell in the dynamic programming table \(f_{c,j}, c \in [0, z_c^U], j \in \{0, 1, \ldots, n - 1\}\), where \(z_c^U = \max(z_c) - |P_k|\), is a pair of values \(q_{c,j}\) and \(l_{c,j}\). \(f_{c,j} = (q_{c,j}, l_{c,j})\), stores information about \(S_{c,j}\). Namely, \(q_{c,j} = |S_{c,j}|\), \(l_{c,j} = |last(S_{c,j})|\) if \(|last(S_{c,j})| \neq \infty \) and \(\infty\) otherwise. We say that \(f_{c,j}/q_{c,j}\) is a dummy (takes a dummy value) iff \(q_{c,j} = \infty\). If \(y_1 = \infty\) and \(y_2 = \infty\) then we assume that they are equal. We introduce a dummy variable \(x_{-1}, D(x_{-1}) = \{0\}\) and a row \(f_{-1,j} = \{1, \ldots, n - 1\}\) to keep uniform notations.

Algorithm 3: Weighted FOCUS(x_0, \ldots, x_{n-1})

1 for \(c \in [-1, z_c^U]\) do
2 \(f_{c,-1} \leftarrow (\infty, \infty)\);
3 \(f_{0,-1} \leftarrow (0, 0)\);
4 for \(j \in [0, n - 1]\) do
5 \(f_{c,j} \leftarrow (\infty, \infty)\);
6 for \(c \in [0, n - 1]\) do
7 if \(x_c \in P_k\) then /* penalizing */
8 if \((k_{-1,j} < 1, len) \) \(\land \) \(q_{c,j} = \infty\) then
9 \(f_{c,j} \leftarrow (q_{c,j} - k_{-1,j}, l_{c,j} + 1)\);
10 else \(f_{c,j} \leftarrow (q_{c,j} - 1, 1)\);
11 if \(x_c \in U_k\) then /* undetermined */
12 if \((k_{-1,j} < 1, len) \land \) \(q_{c,j} = \infty\) then
13 \(f_{c,j} \leftarrow (q_{c,j} - 1, l_{c,j})\);
14 \(f_{c,j} \leftarrow (q_{c,j} - 1, 1)\) else
15 \(f_{c,j} \leftarrow (q_{c,j} - 1, 1)\); /* neutral */
16 return \(f\).

Algorithm 3 gives pseudocode for the propagator. The intuition behind the algorithm is as follows. Again, by cost we mean the number of covered variables in \(U_k\).

If \(x_j \in P_k\) then we do not increase the cost of \(S_{c,j}\) compared to \(S_{c-1,j-1}\) as the cost only depends on \(x_j \in U_k\). Hence, the best move for us is to extend \(last(S_{c,j-1})\) or start a new sequence if it is possible. This is encoded in lines 7 and 11 of the algorithm. Figure 3(a) gives a schematic representation of these arguments.

Table 1: An execution of Algorithm 3 on WEIGHTEDFOCUS from Example 2. Dummy values \(f_{c,j}\) are removed.

If \(x_j \in U_k\) then we do not increase the cost of \(S_{c,j}\). Consider \(S_{c,j-1}\) by increasing cost by \(last(S_{c,j-1})\). This is encoded in lines 12 of the algorithm (Figure 3(b)).

First we prove a property of the dynamic programming table. We define a comparison operation between \(f_{c,j}\) and \(f_{c',j'}\) induced by a comparison operation between \(S_{c,j}\) and \(S_{c',j'}\). \(f_{c,j} \leq f_{c',j'}\) if \((q_{c,j} < q_{c',j'})\) or \(q_{c,j} = q_{c',j'}\) and \(l_{c,j} \leq l_{c',j'}\). In other words, as in a comparison operation between sets, we compare by the cardinality of sequences, \(|S_{c,j}|\) and \(|S_{c',j'}|\), and, by the length of the last sequence in each set, \(last(S_{c,j})\) and \(last(S_{c',j'})\). See Narodytska et al. [2013] for the proofs of the next two lemmas.

Lemma 5. Consider WEIGHTEDFOCUS(X, y_c, len, k, z_c). Let \(f\) be dynamic programming table returned by Algorithm 3. Non-dummy elements \(f_{c,j}\) are monotonically non-increasing in each column, so \(f_{c,j} \leq f_{c',j'}\) \(0 \leq c < c' \leq z_c^U\), \(j = 0, \ldots, n - 1\).

Lemma 6. Consider WEIGHTEDFOCUS(X, y_c, len, k, z_c). The dynamic programming table \(f_{c,j} = (q_{c,j}, l_{c,j})\) \(c \in [0, z_c^U]\), \(j = 0, \ldots, n - 1\), is correct in the sense that if \(f_{c,j}\) exists and it is non-dummy then a corresponding set of sequences \(S_{c,j}\) exists and satisfies conditions \([1, 2]\). The time complexity of Algorithm 3 is \(O(n \max(z_c))\).

Example 4. Table 2 shows an execution of Algorithm 3 on WEIGHTEDFOCUS from Example 2. Note that \(|P_1| = 5\). Hence, \(z_c^U = \max(z_c) - |P_1| = 2\). As can be seen from the table, the constraint has a solution as there exists a set \(S_{2,7} = \{S_{0,3}, S_{0,7}\}\) such that \(|S_{2,7}| = 2\).

Bounds consistency To enforce BC on variables \(x\), we compute an additional DP table \(b_{c,j}, c \in [0, z_c^U]\), \(j \in [-1, n - 1]\) on the reverse sequence of variables \(x\).

Lemma 7. Consider WEIGHTEDFOCUS(X, y_c, len, k, z_c). Bounds consistency can be enforced in \(O(n \max(z_c))\) time.

Proof. (Sketch) We build dynamic programming tables \(f\) and \(b\). We will show that to check if \(x_i = v\) has a support it is sufficient to examine \(O(z_c^U)\) pairs of values \(f_{c_1,i-1}\) and \(b_{c_2,n-i-2}\), \(c_1, c_2 \in [0, z_c^U]\), which are neighbor columns to the \(i\)th column. It is easy to show that if we consider all possible pairs of elements in \(f_{c_1,i-1}\) and \(b_{c_2,n-i-2}\) then we determine if there exists a support for \(x_i = v\). There are \(O(z_c^U \times z_c^U)\) such pairs. The main part of the proof shows that it sufficient to consider \(O(z_c^U)\) such pairs. In particular, to check a support for a variable-value pair \(x_i = v, v > k\), for each \(f_{c_1,i-1}\)
it is sufficient to consider only one element \( b_{c_2, n-i-2} \) such that \( b_{c_2, n-i-2} \) is non-dummy and \( c_2 \) is the maximum value that satisfies inequality \( c_1 + c_2 + 1 \leq z_c^U \). To check a support for a variable-value pair \( x_i = v, v \leq k \), for each \( f_{c_1, i-1} \) it is sufficient to consider only one element \( b_{c_2, n-i-2} \) such that \( b_{c_2, n-i-2} \) is non-dummy and \( c_2 \) is the maximum value that satisfies inequality \( c_1 + c_2 \leq z_c^U \).

We observe a useful property of the constraint. If there exists \( f_{c, n-1} \) such that \( c < \max(z_c) \) and \( z_{c, n-1} < \max(y_c) \) then the constraint is BC. This follows from the observation that given a solution of the constraint \( S_X \), changing a variable value can increase \( \text{cst}(S_X) \) and \( |S_X| \) by at most one.

Alternatively we can decompose \( \text{WEIGHTEDFOCUS} \) using \( O(n) \) additional variables and constraints.

**Proposition 4.** Given \( \text{FOCUS}(X, y_c, \{l, k\}) \), let \( z_c \) be a variable and \( B = \{b_0, b_1, \ldots, b_{n-1}\} \) be a set of variables such that \( \forall b \in B \), \( D(b_i) = \{0, 1\} \). \( \text{WEIGHTEDFOCUS}(X, y_c, \{l, k\}) \) \( \Rightarrow \) \( \text{FOCUS}(X, y_c, \{l, k\}) \) \( \land \) \((\forall i, 0 \leq i < n, (\{x_i \leq k\} \land (b_i = 0)) \lor ((x_i > k) \land (b_i = 1))) \land \sum_{i \in \{0, 1, \ldots, n-1\}} b_i \leq z_c.

Enforcing BC on each constraint of the decomposition is weaker than BC on \( \text{WEIGHTEDFOCUS} \). Given \( x_i \in X \), a value may have a unique support for \( \text{FOCUS} \) which satisfies \( \sum_{i \in \{0, 1, \ldots, n-1\}} b_i \leq z_c \), and conversely. Consider \( n=5 \), \( x_0 = x_2 = 1, x_3 = 0 \), and \( D(x_1) = D(x_4) = \{0, 1\} \), \( y_c = 2 \), \( z_c = 3 \), \( k = 0 \) and \( \text{len} = 3 \). Value 1 for \( x_4 \) corresponds to this case.

### 4 Weighted Springy FOCUS

We consider a further generalization of the FOCUS constraint that combines \( \text{SPRINGYFOCUS} \) and \( \text{WEIGHTEDFOCUS} \). We prove that we can propagate this constraint in \( O(n \max(z_c)) \) time, which is same as enforcing BC on \( \text{WEIGHTEDFOCUS} \).

**Definition 6.** Let \( y_c \) and \( z_c \) be two variables and \( k, \text{len}, h \) be three integers, such that \( 1 \leq \text{len} \leq |X| \) and \( 0 < h < \text{len} - 1 \). An instantiation of \( X \cup \{y_c, z_c\} \) satisfies \( \text{WEIGHTEDSPRINGYFOCUS}(X, y_c, \{l, k\}) \) if there exists a set \( S_X \) of disjoint sequences of indices \( s_{i,j} \) such that five conditions are all satisfied:

1. \( |S_X| \leq y_c \)
2. \( \forall x_i \in X \), \( x_i > k \Rightarrow \exists s_{i,j} \in S_X \) such that \( i \in s_{i,j} \) and \( \sum_{j \in s_{i,j}} x_{j,i} \leq h \)
3. \( \forall i \in \{0, \ldots, \text{len}\} \), \( \forall j, k \leq j \leq i \)
4. \( \sum_{i \in \{0, \ldots, \text{len}\}} s_{i,j} \leq |X| \)
5. \( \text{cst}(s_{i,j}) = \sum_{i \in s_{i,j}} |P_k| \).

We can again partition cost of \( S \) into two terms.

\[
\sum_{s_{i,j} \in S} |s_{i,j}| = \sum_{s_{i,j} \in S} \text{cst}(s_{i,j}) + |P_k|.
\]

The propagator is again based on a dynamic program that for each prefix of variables \( \{x_0, x_1, \ldots, x_j\} \) and given cost \( c \) computes a cover \( S_{c,j} \) of minimum cardinality that covers all penalized variables in the prefix \( \{x_0, x_1, \ldots, x_j\} \) and has cost exactly \( c \). We face the same problem of how to compare two sets \( S_{c,j}^1 \) and \( S_{c,j}^2 \) of minimum cardinality. The issue here is how to compare \( \text{last}(S_{c,j}^1) \) and \( \text{last}(S_{c,j}^2) \) if they cover a different number of neutral variables. Luckily, we can avoid this problem due to the following monotonicity property. If \( \text{last}(S_{c,j}^1) \) and \( \text{last}(S_{c,j}^2) \) are not equal to infinity then they both end at the same position \( j \). Hence, if \( \text{last}(S_{c,j}^1) \leq \text{last}(S_{c,j}^2) \) then the number of neutral variables covered by \( \text{last}(S_{c,j}^1) \) is no larger than the number of neutral variables covered by \( \text{last}(S_{c,j}^2) \). Therefore, we can define order on sets \( S_{c,j} \) as we did in Section 3.5 for \( \text{WEIGHTEDFOCUS} \).

Our bounds disentailment detection algorithm for \( \text{WEIGHTSPRINGYFOCUS} \) mimics Algorithm 3. We omit the pseudocode due to space limitations but highlight two not-trivial differences between this algorithm and Algorithm 3. The first difference is that each cell in the dynamic programming table \( f_{c,j}, c \in [0, z_c^U], j \in [0, 1, \ldots, n - 1] \), where \( z_c^U = \max(z_c) - |F_k| \), is a tripal of values \( q_{c,j}, l_{c,j} \) and \( h_{c,j} \).

Alternatively we can obtain \( S_{c,j} \) from \( S_{c-1,j} \) by increasing \( \text{cst}(S_{c-1,j}) \) by one and increasing the number of covered neutral variables by \( \text{last}(S_{c-1,j}) \). BC can enforced using two modifications of the corresponding algorithm for \( \text{WEIGHTEDFOCUS} \) (a proof is given in Narodytska et al., 2013).

** Lemma 8.** Consider \( \text{WEIGHTEDSPRINGYFOCUS}(X, y_c, \{l, k\}) \). BC can be enforced in \( O(n \max(z_c)) \) time.

**WEIGHTEDSPRINGYFOCUS** can be encoded using the cost-\textit{REGULAR} constraint. The automaton needse 3 counters to compute \( l, k \), and \( h \). Hence, the time complexity of this encoding is \( O(n^3) \). This automaton is non-deterministic as on seeing \( v \leq k \), it either covers the variable or interrupts the last sequence. Unfortunately the non-deterministic cost-\textit{REGULAR} is not implemented in any constraint solver to our knowledge. In contrast, our algorithm takes just \( O(n^2) \) time.

WEIGHTEDSPRINGYFOCUS can also be decomposed using the GCC constraint [Regin, 1996]. We define the following variables for all \( i \in [0, \max(y_c) - 1] \) and \( j \in [0, n - 1] \):

1. \( S_i = \{0, \ldots, \max(y_c)\} \); \( E_i \), the end of the \( i \)th sub-sequence.
2. \( D(E_i) = \{0, \ldots, \max(y_c)\} \);
3. \( T_j \), the index of the subsequence in \( S_X \) containing \( x_j \).
4. \( D(T_j) = \{0, \ldots, \max(y_c)\} \); \( Z_j \), the index of the subsequence in \( S_X \) containing \( x_j \).
5. \( \text{last}(S_X, \{0, \ldots, h\}) \) variables having \( \{0, \ldots, h\} \) as domains.

**5 Experiments**

We used the Choco-2.1.5 solver on an IntelXeon 2.27GHz for the first benchmarks and IntelXeon 3.20GHz for last ones,
both under Linux. We compared the propagators (denoted by F) of WEIGHTEDFOCUS and WEIGHTEDSPRINGYFOCUS against two decompositions (denoted by D1 and D2), using the same search strategies, on three different benchmarks. The first decomposition, restricted to WEIGHTEDFOCUS, is shown in proposition 4 while the second one is shown in Section 4. In the tables, we report for each set the total number of solved instances (#n), then we average both the number of backtracks (#b) and the resolution time (T) in seconds.

**Sports league scheduling (SLS).** We extend a single round-robin problem with \( n = 2p \) teams. Each week each team plays a game either at home or away. Each team plays exactly once all the other teams during a season. We minimize the number of breaks (a break for one team is two consecutive home or two consecutive away games), while fixed weights in \( \{0, 1\} \) are assigned to all games: games with weight 1 are important for TV channels. The goal is to group consecutive weekends where at least one game is important (sum of weights \( > 0 \)), to increase the price of TV broadcast packages. Packages are limited to 5 weeks and should be as short as possible. Table 2 shows results with 16 and 20 teams, on sets of 50 instances with 10 random important games and a limit of 400K backtracks. max\( (y_e) = 3 \) and we search for one solution with \( h \leq 7 \) (instances \( n=1 \)), \( h \leq 6 \) (n-2) and \( h \leq 5 \) (n-3). In our model, inverse-channeling and ALLODDIFFERENT constraints with the strongest propagation level express that each team plays once against each other team. We assign first the constraints. The goal is to build a schedule for rentals that satisfies all demand constraints and minimizes simultaneously the number of rental periods and their total length. We build a Pareto frontier over two cost variables, as Figure 4 shows for one of the instances of this problem.

Table 2: SLS with WEIGHTEDFOCUS and its decomposition.

<table>
<thead>
<tr>
<th>( n=16 )</th>
<th>( n=16 )</th>
<th>( n=16 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n</td>
<td>#b</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>50</td>
<td>0.9K</td>
</tr>
<tr>
<td>D1</td>
<td>50</td>
<td>3.4K</td>
</tr>
<tr>
<td>D2</td>
<td>50</td>
<td>3.4K</td>
</tr>
</tbody>
</table>

Table 3: Scheduling with Rentals.

<table>
<thead>
<tr>
<th>h</th>
<th>#n</th>
<th>#b</th>
<th>T</th>
<th>#n</th>
<th>#b</th>
<th>T</th>
<th>#n</th>
<th>#b</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>0</td>
<td>20</td>
<td>349K</td>
<td>20</td>
<td>1M</td>
<td>192.2</td>
<td>20</td>
<td>1M</td>
<td>233.7</td>
</tr>
<tr>
<td>D1</td>
<td>20</td>
<td>529K</td>
<td>74.7</td>
<td>20</td>
<td>1M</td>
<td>251.2</td>
<td>20</td>
<td>1M</td>
<td>325.6</td>
</tr>
<tr>
<td>D2</td>
<td>20</td>
<td>529K</td>
<td>74.7</td>
<td>20</td>
<td>1M</td>
<td>251.2</td>
<td>20</td>
<td>1M</td>
<td>325.6</td>
</tr>
</tbody>
</table>

Table 4: Sorting Chords.

<table>
<thead>
<tr>
<th>h</th>
<th>#n</th>
<th>#b</th>
<th>T</th>
<th>#n</th>
<th>#b</th>
<th>T</th>
<th>#n</th>
<th>#b</th>
<th>T</th>
<th>#n</th>
<th>#b</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>0</td>
<td>19</td>
<td>3M</td>
<td>154.5</td>
<td>18</td>
<td>2M</td>
<td>553.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>20</td>
<td>2M</td>
<td>396.8</td>
<td>17</td>
<td>3M</td>
<td>660</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>20</td>
<td>2M</td>
<td>396.8</td>
<td>17</td>
<td>3M</td>
<td>660</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Cumulative Scheduling with Rentals.** Given a horizon of \( n \) days and a set of time intervals \([s_i, e_i], i \in \{1, 2, \ldots, p\}\), a company needs to rent a machine between \( l_i \) and \( u_i \) times within each time interval \([s_i, e_i]\). We assume that the cost of the rental period is proportional to its length. On top of this, each time the machine is rented we pay a fixed cost. The problem is then defined as a conjunction of one \( \text{WEIGHTEDSPRINGYFOCUS}(X, y_e, \text{len}, h, 0, z_e) \) with a set of \( \text{AMONG} \) constraints. The goal is to build a schedule for rentals that satisfies all demand constraints and minimizes simultaneously the number of rental periods and their total length. We build a Pareto frontier over two cost variables, as Figure 4 shows for one of the instances of this problem.

Figure 4: Pareto frontier for Scheduling with Rentals.
References


