Constraints (2016) 21:495–532
DOI 10.1007/s10601-015-9233-7

Three generalizations of the FOCUS constraint

Nina Narodytska[1] · Thierry Petit[2,3] · Mohamed Siala[4] · Toby Walsh[5]

Published online: 1 December 2015

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. Moreover, we propose ILP and CSP decompositions.

Keywords Constraint propagation · Global constraints

✉ Mohamed Siala
mohamed.siala@insight-centre.org

Nina Narodytska
nina.n@samsung.com

Thierry Petit
tpetit@wpi.edu

Toby Walsh
toby.walsh@nicta.com.au

[1] Samsung Research America, Mountain View, CA, USA

[2] School of Business, Worcester Polytechnic Institute, Worcester, MA, USA

[3] Mines Nantes, Nantes, France

[4] Insight Centre for Data Analytics, Department of Computer Science, University College Cork, Cork, Ireland

[5] UNSW and Data61 (formerly NICTA), Sydney, Australia
Now at TU Berlin funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under AMPLIFY 670077.

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 paradigm to express and filter such constraints. In particular, several constraints have been proposed for obtaining well-balanced solutions [9, 11, 17]. Recently, the FOCUS constraint [12] 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 integer variables and $s_{i,j}$ is a sequence of indices of consecutive variables in X, such that $s_{i,j} = [i, i+1, \ldots, j]$, $0 \leq i \leq j < n$. For each variable x, we denote by $D(x)$ the domain of x and finally, we let $|E|$ be the size of a collection E.

Definition 1 ([12]) Let y_c be a variable. Let k and *len* be two integers, $1 \leq len \leq |X|$. An instantiation of $X \cup \{y_c\}$ satisfies FOCUS (X, y_c, len, k) iff 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. $\forall x_l \in X, x_l > k \Leftrightarrow \exists s_{i,j} \in S_X$ such that $l \in s_{i,j}$
3. $\forall s_{i,j} \in S_X, j - i + 1 \leq len$

Example 1 Let $k = 0$, $D(y_c) = \{2\}$, $X = [x_0, .., x_5]$, $D(x_0) = \{1\}$, $D(x_1) = \{3\}$, $D(x_2) = \{1\}$, $D(x_3) = \{0\}$, $D(x_4) = \{1\}$, $D(x_5) = \{0\}$. If $len = 6$, then FOCUS (X, y_c, len, k) is satisfied since we can have 2 disjoint sequences of length ≤ 6 of consecutive variables with a value strictly positive, *i.e.*, $\langle x_0, x_1, x_2 \rangle$, and $\langle x_4 \rangle$. If $len = 2$, FOCUS (X, y_c, len, k) becomes violated since it is impossible to include all the strictly positive variables in X with only 2 sequences of length ≤ 2.

FOCUS can be used in various contexts including cumulative scheduling problems where some excesses of capacity can be tolerated to obtain a solution [12]. 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 [4]. 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 *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*.

Consider a simple example of a resource R with a capacity equal to 3. We use a sequence of variables $[x_0, .., x_9]$ to model the amount of consumed capacity at time unit i (e.g., one day). Suppose that some activities are already scheduled on R such that the current assignment of $[x_0, .., x_9]$ is:

$$[x_0, .., x_9]: \quad 4 \quad 2 \quad 4 \quad 2 \quad 2 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0$$

In this example, the first day requires a capacity equal to 4, the second requires 2, etc. The standard capacity constraints are exceeded in x_0 and x_2.

Suppose that an additional activity has to be scheduled on this resource. The new activity has a duration of 5 days, each of which consumes 2 units of capacity. The followings sequence (denoted S_1) shows the new resource consumption if we start the new activity at x_1. The bold values show the new capacity requirement after adding the new activity.

$$[x_0, .., x_9] \quad 4 \quad \mathbf{4} \quad \mathbf{6} \quad \mathbf{4} \quad \mathbf{4} \quad \mathbf{2} \quad 0 \quad 0 \quad 0 \quad 0$$

The new sequence S_1 satisfies FOCUS $(X, [1, 1], 5, 3)$ since we have only one subsequence where the capacity constraints are all exceeded (i.e. $\langle x_0, x_1, x_2, x_3, x_4 \rangle$). However, there is no possible way to satisfy the constraint if the length is equal to 3. FOCUS $(X, [1, 1], 3, 3)$ is violated.

Consider now a form of relaxation by allowing some variables in the sub-sequences to have values that do not exceed capacity. In this case, a solution is possible if we start the additional activity at x_5 (denoted S_2). That is:

$$[x_0, .., x_9]: \quad 4 \quad 2 \quad 4 \quad 2 \quad 2 \quad \mathbf{2} \quad \mathbf{2} \quad \mathbf{2} \quad \mathbf{2} \quad \mathbf{2}$$

The unique subsequence in S_2 where some capacity constraints are exceeded is $\langle x_0, x_1, x_2 \rangle$. Relaxing FOCUS in this sense might be very useful in practice.

Consider now again FOCUS $(X, [2, 2], 5, 3)$. The two solutions S_1 and S_2 satisfy the constraint. Notice that there is 6 capacity excesses in S_1 (i.e., in x_0, x_1, x_2, x_3, x_4) and only 2 in S_2 (i.e., in x_0 and x_2). Therefore, one might prefer S_2 since we have less capacity excesses although the project ends later. Restricting the length subsequences to be at most 2 in this example will prune the first solution.

We tackle those issues in this paper by means of three generalizations of FOCUS. SPRINGYFOCUS tolerates within each sequence $s_{i,j} \in S_X$ 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. SPRINGYFOCUS 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. Propagating such constraints, i.e. complementary to an objective function, is well-known to be important [10, 18]. We present and experiment with filtering algorithms and decompositions therefore for each constraint. One of the decompositions highlights a relation between SPRINGYFOCUS and a tractable Integer Linear Programming (ILP) problem.

The rest of this paper is organized as follows : We give in Section 2 a short background on Constraint Programming and Network Flows. Next, in Sections 3, 4 and 5, we present

three generations of the FOCUS constraint (denoted by SPRINGYFOCUS, WEIGHTEDFOCUS, and WEIGHTEDSPRINGYFOCUS respectively). In particular, we provide complete filtering algorithms as well as ILP formulations and CSP decompositions. Finally, we evaluate, in Section 6, the impact of the new filtering compared to decompositions.

2 Background

A constraint satisfaction problem (CSP) is defined by a set of variables, each with a finite domain of values, and a set of constraints specifying allowed combinations of values for subsets of variables. For each variable x, we denote by $\min(x)$ (respectively $\max(x)$) the minimum (respectively maximum) value in $D(x)$. Given a constraint C, we denote by *Scope(C)* the set of variables constrained by C. A solution is an assignment of values to the variables satisfying the constraints.

Constraint solvers typically explore partial assignments enforcing a local consistency property using either specialized or general purpose *filtering algorithms* [16]. A filtering algorithm (called also a propagator) is usually associated with one constraint, to remove values that cannot belong to an assignment satisfying this constraint. A local consistency formally characterizes the impact of filtering algorithms. The two most used local consistencies are *domain consistency* (DC) and *bound consistency* (BC). A *support* for a constraint C is a tuple that assigns a value to each variable in *Scope(C)* from its domain which satisfies C. A *bounds support* for a constraint C is a tuple that assigns a value to each variable in *Scope(C)* which is between the maximum and minimum in its domain which satisfies C. A constraint C is *domain consistent* (DC) if and only if for each variable $x_i \in$*Scope(C)*, every value in the current domain of x_i belongs to a *support*. A constraint C is *bounds consistent* (BC) if and only if for each variable $x_i \in$*Scope(C)*, there is a *bounds support* for the maximum and minimum value in its current domain. A *CSP* is DC/BC if and only if each constraint is DC/BC. Regarding FOCUS, a complete filtering algorithm (i.e. achieving *domain consistency*) is proposed in [12] running in $O(n)$ time complexity.

A *flow network* is a weighted directed graph $G = (V, E)$ where each edge e has a capacity between non-negative integers $l(e)$ and $u(e)$, and an integer cost $w(e)$. A *feasible flow* in a flow network between a source (s) and a sink (t), (s,t)-flow, is a function $f : E \rightarrow \mathbb{Z}^+$ satisfying two conditions: $f(e) \in [l(e), u(e)]$, $\forall e \in E$ and the *flow conservation* law that ensures that the amount of incoming flow should be equal to the amount of outgoing flow for all nodes except the source and the sink. The *value* of a (s,t)-flow is the amount of flow leaving the sink s. The *cost* of a flow f is $w(f) = \sum_{e \in E} w(e) f(e)$. A *minimum cost flow* is a feasible flow with the minimum cost [1].

3 Springy FOCUS

3.1 Definition

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 (depicted in Fig. 1) for a given resource of capacity 3. Each variable $x_i \in X$ represents the resource consumption

and is defined per unit of time (e.g., one day). Initially, 4 activities are fixed (drawing A) as follows:

1. Activity 1 starts at day 0 and requires 4 units of capacity during one day
2. Activity 2 starts at day 1 and requires 2 units of capacity during one day
3. Activity 3 starts at day 2 and requires 4 units of capacity during one day
4. Activity 4 starts at day 3 and requires 2 units of capacity during two days

Suppose now that an additional activity with 2 units of capacity and a duration of 5 days remains to be scheduled. Suppose also that the domain of the starting time of the new activity is $D(st) = [1, 5]$. If FOCUS$(X, y_c = 1, 5, 3)$ is imposed then this activity 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 $len = 4$ 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, h be three integers, $1 \leq len \leq |X|$, $0 \leq h < len - 1$. An instantiation of $X \cup \{y_c\}$ satisfies SPRINGYFOCUS (X, y_c, len, h, k) 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. $\forall x_l \in X, x_l > k \Rightarrow \exists s_{i,j} \in S_X$ such that $l \in s_{i,j}$
3. $\forall s_{i,j} \in S_X, j - i + 1 \leq len, x_i > k$ and $x_j > k$.
4. $\forall s_{i,j} \in S_X, |\{l \in s_{i,j}, x_l \leq k\}| \leq h$

3.2 Filtering algorithm

Bounds consistency (BC) on SPRINGYFOCUS is equivalent to *domain consistency*: any solution can be turned into a solution that only uses the lower bound $\min(x_l)$ or the upper

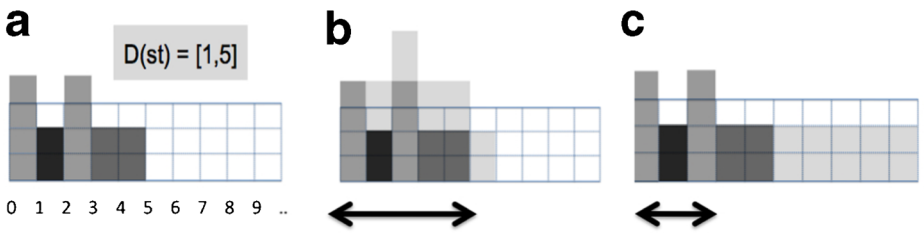

Fig. 1 Introducing SPRINGYFOCUS Example of a resource with capacity equal to 3. Each day is represented by one unit in the horizontal axis. The capacity usage is represented by the vertical axis. **A** Problem with 4 fixed activities: activity 1 scheduled on day 0 with 4 units of capacity; activity 2 scheduled on day 1 with 2 units of capacity; activity 3 scheduled on day 2 with 4 units of capacity; and activity 4 scheduled on days 3 and 4 with 2 units of capacity each. An additional activity of length 5 should start from time 1 to 5 (i.e. the domain of the starting time of the new activity is D(st)=[1,5]). **B** Solution satisfying FOCUS$(X, [1, 1], 5, 3)$, with a new machine rented for 5 days. **C** Practical solution violating FOCUS$(X, [1, 1], 5, 3)$, with a new machine rented for 3 days but not used on the second day

bound $\max(x_l)$ of the domain $D(x_l)$ of each $x_l \in X$ (this observation was made for FOCUS [12]). 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 give a formal definition.

Definition 3 Focus cardinality
Let X be a sequence of variables subject to SPRINGYFOCUS(X, y_c, len, h, k). The focus cardinality of any subsequence $s \subset X$, denoted $fc(s)$, is defined as follows:

$$fc(s) = \min_{\omega \in D(y_c)} \text{SPRINGYFOCUS}(s, y_c^{\omega}, len, h, k) \text{is satisfiable} \mid D(y_c^{\omega}) = \{\omega\}$$

Definition 4 Given $x_l \in X$, we consider three quantities.

1. $\underline{p}(x_l, v_{\leq})$ is the focus cardinality of $[x_0, x_1, \ldots, x_l]$, assuming $x_l \leq k$, and $\forall s_{i,j} \in S_{[x_0, x_1, \ldots, x_l]}$, $j \neq l$.
2. $\underline{p_S}(x_l, v_{\leq})$, $0 < l < n-1$, is the focus cardinality of $[x_0, x_1, \ldots, x_j]$, where $l < j < n$, assuming $x_l \leq k$ and $\exists i, 0 \leq i < l, s_{i,j} \in S_{[x_0, x_1, \ldots, x_j]}$. $\underline{p_S}(x_0, v_{\leq}) = \underline{p_S}(x_{n-1}, v_{\leq}) = n + 1$.
3. $\underline{p}(x_l, v_{>})$ is the focus cardinality of $[x_0, x_1, \ldots, x_l]$ assuming $x_l > k$.

Any quantity is equal to $n+1$ if the domain $D(x_l)$ of x_l makes impossible the considered assumption.

We shall use the above notations throughout the paper.

Property 1 $fc(X) = \min(\underline{p}(x_{n-1}, v_{\leq}), \underline{p}(x_{n-1}, v_{>}))$.

Proof By construction from Definitions 2 and 4. □

To compute the quantities of Definition 4 for $x_l \in X$ we use two additional measures.

Definition 5 $\underline{plen}(x_l)$ is the minimum length of a sequence in $S_{[x_0, x_1, \ldots, x_l]}$ containing x_l among instantiations of $[x_0, x_1, \ldots, x_l]$ where the number of sequences is $fc([x_0, x_1, \ldots, x_l])$. $\underline{plen}(x_l) = 0$ if $\forall s_{i,j} \in S_{[x_0, x_1, \ldots, x_l]}$, $j \neq l$.

Definition 6 $\underline{card}(x_l)$ is the minimum number of values $v \leq k$ in the current sequence in $S_{[x_0, x_1, \ldots, x_l]}$, equal to 0 if $\forall s_{i,j} \in S_{[x_0, x_1, \ldots, x_l]}$, $j \neq l$. $\underline{card}(x_l)$ assumes that $x_l > k$. It has to be decreased it by one if $x_l \leq k$.

Proofs of following recursive Lemmas 1 to 4 omit the obvious cases where quantities take the default value $n + 1$.

Lemma 1 (initialization) $\underline{p}(x_0, v_{\leq}) = 0$ *if* $\min(x_0) \leq k$, *and* $n+1$ *otherwise;* $\underline{p_S}(x_0, v_{\leq}) = n + 1$*;* $\underline{p}(x_0, v_{>}) = 1$ *if* $\max(x_0) > k$ *and* $n + 1$ *otherwise;* $\underline{plen}(x_0) = 1$ *if* $\max(x_0) > k$ *and 0 otherwise;* $\underline{card}(x_0) = 0$.

Proof From item 4 of Definition 2, a sequence in S_X cannot start with a value $v \leq k$. Thus, $\underline{p_S}(x_0, v_{\leq}) = n + 1$ and $\underline{card}(x_0) = 0$. If x_0 can take a value $v > k$ then by Definition 4, $\underline{p}(x_0, v_{>}) = 1$ and $\underline{plen}(x_0) = 1$. □

We now consider a variable $x_l \in X, 0 < l < n$.

Lemma 2 ($\underline{p}(x_l, v_{\leq})$) *If* $\min(x_l) \leq k$ *then* $\underline{p}(x_l, v_{\leq}) = \min(\underline{p}(x_{l-1}, v_{\leq}), \underline{p}(x_{l-1}, v_{>}))$, *else* $\underline{p}(x_l, v_{\leq}) = n + 1$.

Proof If $\min(x_l) \leq k$ then $\underline{p_S}(x_{l-1}, v_{\leq})$ must not be considered: it would imply that a sequence in S_X ends by a value $v \leq k$ for x_{l-1}. From Property 1, the focus cardinality of the previous sequence is $\min(\underline{p}(x_{l-1}, v_{\leq}), \underline{p}(x_{l-1}, v_{>}))$. □

Lemma 3 ($\underline{p_S}(x_l, v_{\leq})$) *If* $\min(x_i) > k$, $\underline{p_S}(x_i, v_{\leq}) = n + 1$. *Otherwise, if* $\underline{plen}(x_{i-1}) \in \{0, len - 1, len\} \vee \underline{card}(x_{i-1}) = h$ *then* $\underline{p_S}(x_i, v_{\leq}) = n + 1$, *else* $\underline{p_S}(x_i, v_{\leq}) = \min(\underline{p_S}(x_{i-1}, v_{\leq}), \underline{p}(x_{i-1}, v_{>}))$.

Proof If $\min(x_i) \leq k$ we have three cases to consider. (1) If either $\underline{plen}(x_{i-1}) = 0$ or $\underline{plen}(x_{i-1}) = len$ then from item 3 of Definition 2 a sequence in S_X cannot start with a value $v_i \leq k$: $\underline{p_S}(x_i, v_{\leq}) = n + 1$. (2) If $\underline{plen}(x_{i-1}) = len - 1$ then from Definition 2 the current variable x_i cannot end the sequence with a value $v_i \leq k$. (3) Otherwise, from item 3 of Definition 2, $\underline{p}(x_{i-1}, v_{\leq})$ is not considered. Thus, from Property 1, $\underline{p_S}(x_i, v_{\leq}) = \min(\underline{p_S}(x_{i-1}, v_{\leq}), \underline{p}(x_{i-1}, v_{>}))$. □

Lemma 4 ($\underline{p}(x_l, v_{>})$) *If* $\max(x_l) \leq k$ *then* $\underline{p}(x_l, v_{>}) = n + 1$. *Otherwise, If* $\underline{plen}(x_{l-1}) \in \{0, len\}$, $\underline{p}(x_l, v_{>}) = \min(\underline{p}(x_{l-1}, v_{>})+1, \underline{p}(x_{l-1}, v_{\leq})+1)$, *else* $\underline{p}(x_l, v_{>}) = \min(\underline{p}(x_{l-1}, v_{>}), \underline{p_S}(x_{l-1}, v_{\leq}), \underline{p}(x_{l-1}, v_{\leq}) + 1)$.

Proof If $\underline{plen}(x_{l-1}) \in \{0, len\}$ a new sequence has to be considered: $\underline{p_S}(x_{l-1}, v_{\leq})$ must not be considered, from item 3 of Definition 2. Thus, $\underline{p}(x_l, v_{>}) = \min(\underline{p}(x_{l-1}, v_{>}) + 1, \underline{p}(x_{l-1}, v_{\leq})+1)$. Otherwise, either a new sequence has to be considered ($\underline{p}(x_{l-1}, v_{\leq})+1$) or the value is equal to the focus cardinality of the previous sequence ending in x_{l-1}. □

Proposition 1 ($\underline{plen}(x_l)$) *If* $\min(\underline{p_S}(x_{l-1}, v_{\leq}), \underline{p}(x_{l-1}, v_{>})) < \underline{p}(x_{l-1}, v_{\leq}) + 1 \wedge \underline{plen}(x_{l-1}) < len$ *then* $\underline{plen}(x_l) = \underline{plen}(x_{l-1}) + 1$. *Otherwise, if* $\underline{p}(x_l, v_{>})) < n + 1$ *then* $\underline{plen}(x_l) = 1$, *else* $\underline{plen}(x_l) = 0$.

Proof By construction from Definition 5 and Lemmas 1,2,3,4. □

Proposition 2 ($\underline{card}(x_l)$) *If* $\underline{plen}(x_l) = 1$ *then* $\underline{card}(x_l) = 0$. *Otherwise, if* $\underline{p}(x_l, v_{>}) = n + 1$ *then* $\underline{card}(x_l) = \underline{card}(x_{l-1}) + 1$, *else* $\underline{card}(x_l) = \underline{card}(x_{l-1})$.

Proof By construction from Definition 5,6 and Lemmas 1 and 4. □

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

```
1  pre := new Integer[|X|][4][] ;
2  for l ∈ 0..n − 1 do
3      pre[l][0] := new Integer[2];
                                                  /* Initialization from Lemma 1 */
4  if min(x_0) ≤ k then
5      pre[0][0][0] := 0 ;
6  else
7      pre[0][0][0] := n + 1 ;
8  pre[0][0][1] := n + 1 ;
9  if max(x_0) > k then
10     pre[0][1] := 1 ;
11 else
12     pre[0][1] := n + 1 ;
13 if max(x_0) > k then
14     pre[0][2] := 1 ;
15 else
16     pre[0][2] := 0 ;
17 pre[0][3] := 0 ;
18 for l ∈ 1..n − 1 do
                                                                        /* Lemma 2 */
19     if min(x_l) ≤ k then
20         pre[l][0][0] := min(pre[l − 1][0][0], pre[l − 1][1]) ;
21     else
22         pre[l][0][0] := n + 1 ;
                                                                        /* Lemma 3 */
23     if min(x_l) > k then
24         pre[l][0][1] := n + 1 ;
25     else
26         if pre[l − 1][2] ∈ {0, len − 1, len} ∨ pre[l − 1][3] = h then
27             pre[l][0][1] := n + 1 ;
28         else
29             pre[l][0][1] := min(pre[l − 1][0][1], pre[i − 1][0][0]) ;
                                                                        /* Lemma 4 */
30     if max(x_l) ≤ k then
31         pre[l][1] := n + 1 ;
32     else
33         if pre[l − 1][2] ∈ {0, len} then
34             pre[l][1] = min(pre[l − 1][1] + 1, pre[l − 1][0][0] + 1) ;
35         else
36             pre[l][1] = min(pre[l − 1][1], pre[l − 1][0][1], pre[l − 1][0][0] + 1)
                                                                /* Proposition 1 */
37     if min (pre[l − 1][0][1], pre[l − 1][1]) < pre[l − 1][0][0] + 1 ∧ pre[l − 1][2] < len then
38         pre[l][2] = pre[l − 1][2] + 1 ;
39     else
40         if pre[l][1] < n + 1 then
41             pre[l][2] := 1 ;
42         else
43             pre[l][2] := 0 ;
                                                                /* Proposition 2 */
44     if pre[l][2] = 1 then
45         pre[l][3] := 0 ;
46     else
47         if pre[l][1] = n + 1 then
48             pre[l][3]) := pre[l − 1] + 1;
49         else
50             pre[l][3]) := pre[l − 1] ;
51 return pre;
```

Algorithm 1 implements the lemmas with $pre[l][0][0] = \underline{p}(x_l, v_{\leq})$, $pre[l][0][1] = \underline{p_S}(x_l, v_{\leq})$, $pre[l][1] = \underline{p}(x_l, v_{>})$, $pre[l][2] = \underline{plen}(x_l)$, $pre[l][3] = \underline{card}(x_l)$.

The principle of Algorithm 2 is the following. First, $lb = fc(X)$ is computed with x_{n-1}. We execute Algorithm 1 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_l. To aggregate them, we implement regret mechanisms directly derived from Propositions 1 and 2,, 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 [12]). Algorithm 2 performs a constant number of traversals of the set X. Its time complexity is $O(n)$, which is optimal.

Algorithm 2: FILTERING(X, y_c, len, k, h): Set of variables

```
1  pre := MinCards(X, len, k, h) ;
2  Integer lb := min(pre[n − 1][0][0], pre[n − 1][1]);
3  if min(y_c) < lb then D(y_c) := D(y_c) \ [min(y_c), lb[  if min(y_c) = max(y_c) then
4      suf := MinCards([x_{n−1}, x_{n−2}, ..., x_0], len, k, h) ;
5      for l ∈ 0..n − 1 do
6          if pre[l][0][0] + suf[n − 1 − l][0][0] > max(y_c) then
7              Integer regret := 0; Integer add := 0;
8              if pre[l][1] ≤ pre[l][0][1] then add := add + 1  if
               suf[n − 1 − l][1] ≤ suf[n − 1 − l][0][1] then add:=add+1  if
               pre[l][2] + suf[n − 1 − l][2] − 1 ≤ len ∧
               pre[l][3] + suf[n − 1 − l][3] + add − 1 ≤ h then regret := 1  if
               pre[l][0][1] + suf[n − 1 − l][0][1] − regret > max(y_c) then D(x_i) :=
               D(x_i)\ [min(x_i), k]
9          Integer regret := 0;
10         if pre[l][2] + suf[n − 1 − l][2] − 1 ≤ len ∧ pre[l][3] + suf[n − 1 − l][3] − 1 ≤ h
           then regret := 1  if pre[l][1] + suf[n − 1 − l][1] − regret > max(y_c) then
11             D(x_i) := D(x_i)\ ]k, max(x_i)];
12 return X ∪ {y_c};
```

3.3 Integer Linear Programming formulation

In this section we present a new Integer Linear Programming (ILP) formulation of SPRINGYFOCUS. This connection highlights a relation between SPRINGYFOCUS and a tractable ILP problem. It adds one more constraint to a bag of constraints that can be propagated using shortest path or network flow reformulations [6, 13, 14].

We first present a bounds disentailment technique. We use the following notations from [12].

Definition 7 ([12]) Given an integer k, a variable $x_l \in X$ is:

- Penalizing, (P_k), iff $\min(x_l) > k$.
- Neutral, (N_k), iff $\max(x_l) \leq k$.
- Undetermined, (U_k), otherwise.

We say $x_l \in P_k$ iff x_l is labelled P_k, and similarly for U_k and N_k.

The main observation behind the reformulation is that we can relax the requirement of disjointness of sequences in S_X (Definition 2) and find a solution of the SPRINGYFOCUS constraint. This solution can be transformed into a solution where sequences in S_X

are disjoint as we can truncate the overlaps. If we drop the requirement of disjointness of sequences in S_X then we only need to consider at most n possible sequences $s_{i,i+len_i-1}$, $i \in \{0, 1, \ldots, n-1\}$, x_i and x_{i+len_i-1} are not neutral, and len_i is the maximal possible length of a sequence that starts at the ith position. Note that len_i does not have to be equal to len as $s_{i,i+len_i-1}$ can cover at most h variables that take values less than or equal to k. We call the set of these sequences S_X^o.

Example 2 Consider $X = [x_0, x_1 \ldots, x_8]$ and SPRINGYFOCUS (X, [3,3], 3, 1, 0) with $D(x_0) = D(x_2) = D(x_5) = D(x_7) = D(x_8) = \{1\}$, $D(x_1) = D(x_3) = D(x_4) = 0$ and $D(x_6) = \{0, 1\}$. There are 9 sequences to consider as there are 9 variables. We have 5 valid sequences that are schematically shown in black in Fig. 2a. Hence, $S_X^o = \{s_{0,2}, s_{5,7}, s_{6,8}, s_{7,8}, s_{8,8}\}$. The remaining 4 sequences, $s_{1,2}$, $s_{2,3}$, $s_{3,3}$ and $s_{4,6}$, are discarded, as a sequence should not start(finish) at a neutral variable. We highlighted invalid sequences in grey.

We denote the SPRINGYFOCUS constraint without the disjointness requirement SPRINGYFOCUSOVERLAP. More formally we define SPRINGYFOCUSOVERLAP as follows.

Definition 8 Let y_c be a variable and k, len, h be three integers, $1 \leq len \leq |X|$, $0 \leq h < len - 1$. An instantiation of $X \cup \{y_c\}$ satisfies SPRINGYFOCUSOVERLAP(X, y_c, len, h, k) iff there exists a set $S_X \subseteq S_X^o$ of sequences *(not necessary disjoint)* of indices $s_{i,j}$ such that four conditions are all satisfied:

1. $|S_X| \leq y_c$
2. $\forall x_l \in X, x_l > k \Rightarrow \exists s_{i,j} \in S_X$ such that $l \in s_{i,j}$
3. $\forall s_{i,j} \in S_X, j - i + 1 \leq len, x_i > k$ and $x_j > k$
4. $\forall s_{i,j} \in S_X, |\{l \in s_{i,j}, x_l \leq k\}| \leq h$

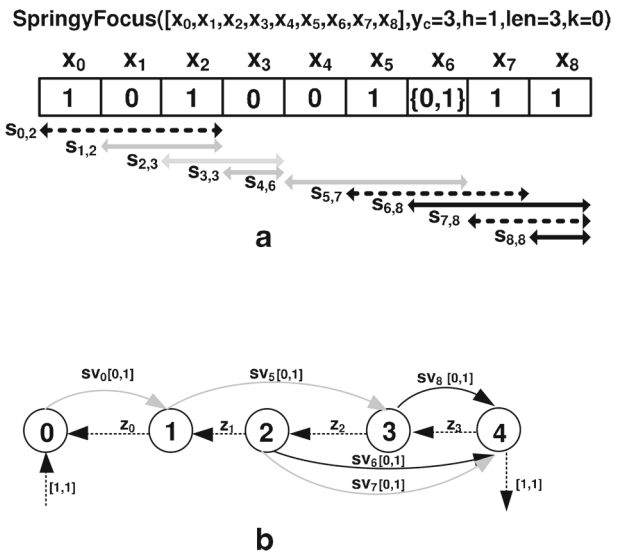

Fig. 2 The set of possible sequences in S_X from Example 2

Lemma 5 SPRINGYFOCUS(X, y_c, len, h, k) *has a solution iff* SPRINGYFOCUS OVERLAP(X, y_c, len, h, k) *has a solution.*

Proof $\Leftarrow$ Let $I[X \cup \{yc\}]$ be a solution of PRINGYFOCUSOVERLA. We order sequences in S_X by their starting points and process them in this order. Let $s_{i,i+len_i-1}$ and $s_{j,j+len_j-1}$ be the first two consecutive sequences in S_X that overlap. We update S_X. First, we remove $s_{j,j+len_j-1}$: $S_X = S_X \setminus \{s_{j,j+len_j-1}\}$. Consider a sequence $s_{i+len_i,j+len_j-1}$. By definition, $x_{j+len_j-1} > k$. If $s_{i+len_i,j+len_j-1}$ has a prefix that contains only neutral variables then we cut it from the sequence and obtain $s_{i',j+len_j-1}$. We add this sequence to our set: $S_X = S_X \cup \{s_{i',j+len_j-1}\}$. So, we cut the prefix of $s_{j,j+len_j-1}$ to avoid the overlap and made sure that the new sequence does not start or end at a neutral variable. This does not change the cardinality $|S_X|$. We continue this procedure for the rest of the sequences. The updated set S_X covers the same set of penalizing variables as the original set and all sequences are disjoint.

$\Rightarrow$ Let $I[X \cup \{y_c\}]$ be a solution of SPRINGYFOCUS. We extend each sequence to its maximal length to the right. This gives a solution of PRINGYFOCUSOVERLAP. □

Example 3 Consider SPRINGYFOCUSOVERLAP from Example 2. $S_X = \{s_{0,2}, s_{5,7}, s_{7,8}\}$ is a possible solution (dashed lines in Fig. 2a). We can cut the prefix of $s_{7,8}$ to avoid an overlap between $s_{5,7}$ and $s_{7,8}$. We obtain $s_{8,8}$ which does not start or finish at a neutral variable. Hence, $S_X = \big(S_X \cup \{s_{8,8}\}\big) \setminus \{s_{7,8}\} = \{s_{0,2}, s_{5,7}, s_{8,8}\}$.

Thanks to Lemma 5 we build an ILP reformulation for SPRINGYFOCUSOVERLAP, solve this ILP and transform to a solution of the SPRINGYFOCUS constraint. We introduce one Boolean variable sv_i for each sequence in S_X^o. We can write an integer linear program:

$$\text{Minimize} \sum_{i:s_{i,i+len_i} \in S_X^o} sv_i \tag{1}$$

$$\sum_{\{sv_i : x_j \in s_{i,i+len_i-1}\}} sv_i \geq 1 \quad \forall x_j \in P_k \tag{2}$$

$$sv_i \in \{0, 1\} \quad \forall sv_i. \tag{3}$$

Lemma 6 SPRINGYFOCUSOVERLAP (X, y_c, len, h, k) *is satisfiable iff the ILP system 1–3 has a solution of cost less than or equal to* $\max(y_c)$.

Proof $\Leftarrow$ Suppose the system described by (1)–(3) has a solution $I[sv]$. We define $S = \{s_{i,i+len_i} | sv_i = 1\}$. Equation 2 ensures that at least one sequence covers a penalizing variable. Equation 1 ensures that the number of selected sequences is at most $\max(y_c)$.

As the rest of uncovered variables in X are undetermined or neutral variables, we can construct an assignment based on S_X. We set all undetermined variables covered by S_X to 1 and all undetermined variables uncovered by S_X to 0. This assignment clearly satisfies SPRINGYFOCUSOVERLAP(X, y_c, len, h, k).

$\Rightarrow$ Suppose there is a solution of the SPRINGYFOCUSOVERLAP(X, y_c, len, h, k) constraint $I[X \cup \{y_c\}]$ and $S_X = \{s_{i_1,j_1}, \ldots, s_{i_p,j_p}\}$ be the set of corresponding sequences. We set variable sv_i to 1 iff $s_{i,i+len_i-1} \in S_X$. This assignment satisfies (1)–(3). □

Next we note that the ILP system (1)–(3) has the consecutive ones properties on columns. This means that the corresponding matrix can be transformed to a network flow matrix

using a procedure described by Veinott and Wagner [19]. We consider the transformation on SPRINGYFOCUS from Example 5. This transformation is similar to the one used to propagate the SEQUENCE constraint [6].

Example 4 Consider SPRINGYFOCUS from Example 2. We build an ILP that corresponds to an equivalent SPRINGYFOCUSOVERLAP constraint using (1)–(3). Note that we do not introduce variables sv_1, sv_2, sv_3 and sv_4 for discarded sequences $s_{1,3}, s_{2,3}, s_{3,3}$ and $s_{4,6}$:

$$\text{Minimize} \sum_{i \in \{0,5,6,7,8\}} sv_i \tag{4}$$

$$sv_0 \geq 1 \tag{5}$$

$$sv_5 \geq 1 \tag{6}$$

$$sv_5 + sv_6 + sv_7 \geq 1, \tag{7}$$

$$sv_6 + sv_7 + sv_8 \geq 1, \tag{8}$$

where $sv_i \in \{0, 1\}$. By introducing surplus/slack variables, z_i, we convert this to a set of equalities:

$$\text{Minimize} \sum_{i \in \{0,5,6,7,8\}} sv_i \tag{9}$$

$$sv_0 - z_0 = 1 \tag{10}$$

$$sv_5 - z_1 = 1 \tag{11}$$

$$sv_5 + sv_6 + sv_7 - z_2 = 1, \tag{12}$$

$$sv_6 + sv_7 + sv_8 - z_3 = 1, \tag{13}$$

In matrix form, this is:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} sv_0 \\ \vdots \\ sv_8 \\ z_0 \\ \vdots \\ z_3 \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

We append a row of zeros to the matrix and subtract the ith row from $i + 1$th row for $i = 1$ to 4. These operations do not change the set of solutions. This gives:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} sv_0 \\ \vdots \\ sv_8 \\ z_0 \\ \vdots \\ z_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \end{pmatrix}$$

The corresponding network flow graph is shown in Fig. 2b. The dashed arcs have cost zero and solid arcs have cost one. Capacities are shown on arcs. We number nodes from 0 to 4 as we have 4 equations in the transformed ILP. We highlighted in grey a possible solution of cost 3. This solution corresponds to the solution from Example 3.

As the right hand side (RHS) of the ILP system (1)–(2) is a unit vector, the RHS in the transformed ILP is a vector $(1, 0, \ldots, 0, -1)$. In other words, we need to consume one unit of flow that enters the first node in the graph and leaves the last node in the graph. Hence, the problem of finding a min cost flow is equivalent to the problem of finding a shortest path in this graph from 0th to mth node, where m is the number of equations in ILP. Moreover, a shortest path can be found in linear time.

Lemma 7 *Let G be a directed graph that corresponds to the* SPRINGY FOCUS(X, y_c, len, h, k). *A shortest path from 0th to mth node can be found in $O(n)$ time.*

Proof We show that there exists a shortest path from 0th to mth node that does not contain arcs $(i+1, i), i \in \{0, 1, \ldots, m-1\}$. We call these arcs backward arcs and call the remaining arcs – forward arcs.

First, we observe that each node in G has an outgoing arc, because the ith node, $i \in \{0, \ldots, m-1\}$ corresponds to the ith penalizing variable in the constraint and a sequence that starts at a penalizing variable is in S^o_X.

Let π be a shortest path from 0 to m node that uses a backward arc. Consider the first occurrence of a sequence of backward arcs in π: $\pi = (0, \ldots, j, i', \ldots, i, g, f, \ldots, m)$, where $i', \ldots, i$ is a path using only backward arcs. As (i, g) is present in G then (i', g'), $g \leq g'$ is present in G. Hence, we can modify the path π to $\pi = (0, \ldots, j, i', g', \pi', f, \ldots, m)$, where (g', π', f) is a path that uses backward arcs to reach f from g' if $(g', f) \notin G$. As the weight π' is 0, the weight of the updated path π is the same as the weight of the original path. Then we apply the same argument to g' and so on.

Hence, we can use a simple greedy algorithm to find the shortest path. We start at the 0th node and select the longest outgoing arc $(0, i)$. In the node i, we again select the longest arc until will reach the mth node. As we know that there exists a shortest path that only uses forward arcs the greedy algorithm is optimal. □

The same ILP reformulation can be done for the FOCUS constraint [12], which is a special case of SPRINGYFOCUS. For these two constraints, we can use such a bounds disentailment procedure to obtain a $O(n^2)$ filtering algorithm by successively applying the program to the two bounds of the domain of each variable in X.

4 Weighted FOCUS

We present WEIGHTEDFOCUS, that extends FOCUS with a variable z_c limiting the the sum of lengths of all the sequences in S_X, i.e., the number of variables *covered* by a sequence in S_X.

4.1 Definition

WEIGHTEDFOCUS distinguishes between solutions that are equivalent with respect to the number of sequences in S_X but not with respect to their length, as Fig. 3 shows.

Definition 9 Let yc and z_c be two integer variables and k, len be two integers, such that $1 \leq len \leq |X|$. An instantiation of $X \cup \{y_c\} \cup \{z_c\}$ satisfies WEIGHTEDFOCUS

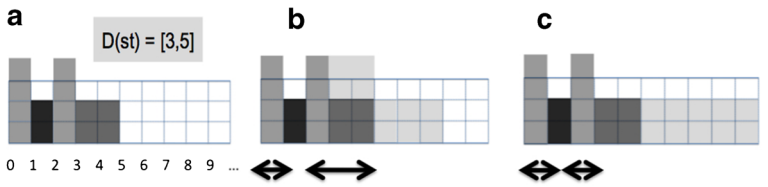

Fig. 3 The same initial configuration of Fig. 1 (**a**) Problem with 4 fixed activities and one activity of length 5 that can start from time 3 to 5 (i.e., D(st)=[3,5]). We assume $D(y_c) = \{2\}$, $len = 3$ and $k = 0$. **b** Solution satisfying WEIGHTEDFOCUS with $z_c = 4$. (C) Solution satisfying WEIGHTEDFOCUS with $z_c = 2$

(X, y_c, 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. $\forall x_l \in X, x_l > k \Leftrightarrow \exists s_{i,j} \in S_X$ such that $l \in s_{i,j}$
3. $\forall s_{i,j} \in S_X, j - i + 1 \leq len$
4. $\sum_{s_{i,j} \in S_X} |s_{i,j}| \leq z_c$.

It should be noted that there are some similarities between WEIGHTEDFOCUS and STRETCH [8]. Indeed given a sequence of variables, the STRETCH constraint restricts the occurrences of consecutive identical values. The particular case of WEIGHTEDFOCUS with Boolean variables is similar to a very specific case of STRETCH with Boolean variables, where only the occurrences of consecutive 1s is bounded. However, STRETCH does not restrict the number of such subsequences. Even though, the semantics behind STRETCH is quite different as the limitation of consecutive values is usually for many values along with many patterns whereas inWEIGHTEDFOCUS the restriction in only for values greater than a threshold. One limitation of WEIGHTEDFOCUS compared to STRETCH is that we do not restrict the minimum size of subsequences with excess. Another limitation is the non-penalization of the extra resource consumption at each unit of time. That is, if $k = 2$, then excess of type $x = 10$ might be very costly compared to two excess of the type $x = 5$.

4.2 Filtering algorithm

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 $cst(s_{i,j})$ is defined as $cst(s_{i,j}) = |\{p | x_p \in U_k, x_p \in s_{i,j}\}|$. We denote cost of S_X, $cst(S_X)$, the sum $cst(S_X) = \sum_{s_{i,j} \in S_X} cst(s_{i,j})$. Given I_X we consider $|P_k| = |\{x_i \in P_k\}|$. We have: $\sum_{s_{i,j} \in S} |s_{i,j}| = \sum_{s_{i,j} \in S} cst(s_{i,j}) + |P_k|$.

We start with explaining the main difficulty in building a propagator for WEIGHTEDFOCUS. The constraint has two optimization variables in its scope (i.e. y_c and z_c) and we might not have a solution that optimizes both variables simultaneously.

Example 5 Consider the set $X = [x_0, x_1, \ldots, x_5]$ with domains $[1, \{0, 1\}, 1, 1, \{0, 1\}, 1]$ and 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 $S_X = \{s_{0,1}, s_{2,3}, s_{5,5}\}$, $y_c = 3$, minimizes $z_c = 4$.

Example 5 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 [3]. 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 6 demonstrates.

Example 6 Consider $X = [x_0, x_1, \ldots, x_7]$ and WEIGHTEDFOCUS $(X, [2, 2], 5, 0, [7, 7])$, 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. $S^1_{1,5} = \{s_{0,2}, s_{3,5}\}$ or $S^2_{1,5} = \{s_{0,4}, s_{5,5}\}$. 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 $S^1_{1,5}$, which is $s_{3,5}$.

Example 6 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 $last(S_{c,j})$ the last sequence in $S_{c,j}$ in this order. If $x_j \in last(S_{c,j})$ then $|last(S_{c,j})|$ is, naturally, the length of $last(S_{c,j})$, otherwise $|last(S_{c,j})| = \infty$.

Ordering of sets $S_{c,j}, c \in [0, \max(z_c)], j \in \{0, 1, \ldots, n-1\}$ We define a comparison operation between two sets $S_{c,j}$ and $S_{c',j'}$:

- $S_{c,j} < S_{c',j'}$ iff $|S_{c,j}| < |S_{c',j'}|$ or $|S_{c,j}| = |S_{c',j'}|$ and $|last(S_{c,j})| < |last(S_{c',j'})|$.
- $S_{c,j} = S_{c',j'}$ iff $|S_{c,j}| = |S_{c',j'}|$ and $|last(S_{c,j})| = |last(S_{c',j'})|$.

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. $cst(S_{c,j}) = c$,
3. $\forall s_{h,g} \in S_{c,j}, x_h \notin U_k$,
4. $S_{c,j}$ *is the first set in the order among all sets that satisfy conditions 1–3.*

As can be seen from definitions above, given a subsequence of variables $x_0, \ldots, x_j$, $S_{c,j}$ is not unique and might 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 7 Consider WEIGHTEDFOCUS from Example 6. Consider the subsequence $[x_0, x_1]$. $S_{0,1} = \{s_{0,0}\}$, $S_{1,1} = \{s_{0,1}\}$. Note that $S_{2,1}$ does not exist. Consider the subsequence $[x_0, \ldots, x_5]$. We have $S_{0,5} = \{s_{0,0}, s_{2,3}, s_{5,5}\}$, $S_{1,5} = \{s_{0,4}, s_{5,5}\}$ and $S_{2,5} = \{s_{0,3}, s_{5,5}\}$. By definition, $last(S_{0,5}) = s_{5,5}$, $last(S_{1,5}) = s_{5,5}$ and $last(S_{2,5}) = s_{5,5}$. Consider the set $S_{1,5}$. Note that there exists another set $S'_{1,5} = \{s_{0,0}, s_{2,5}\}$ that satisfies conditions 1–3. 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 ∞ otherwise. We say that $f_{c,j}/q_{c,j}/l_{c,j}$ is a dummy (takes a dummy value) iff $f_{c,j} = \{\infty, \infty\}/q_{c,j} = \infty/l_{c,j} = \infty$. 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 $f_{-1,j}$, $j = -1, \ldots, n-1$ to keep uniform notations.

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

```
1  for c ∈ −1..z_c^U do
2      for j ∈ −1..n − 1 do
3          f_{c,j} ← {∞, ∞};
4  f_{0,−1} ← {0, 0} ;
5  for j ∈ 0..n − 1 do
6      for c ∈ 0..j do
7          if x_j ∈ P_k then                                   /* penalizing */
8              if (l_{c,j−1} ∈ [1, len)) ∨ (q_{c,j−1} = ∞) then
9                  f_{c,j} ← {q_{c,j−1}, l_{c,j−1} + 1};
10             else f_{c,j} ← {q_{c,j−1} + 1, 1}
11         if x_j ∈ U_k then                                 /* undetermined */
12             if (l_{c−1,j−1} ∈ [1, len) ∧ q_{c−1,j−1} = q_{c,j−1}) ∨ (q_{c,j−1} = ∞) then
               f_{c,j} ← {q_{c−1,j−1}, l_{c−1,j−1} + 1} else f_{c,j} ← {q_{c,j−1}, ∞}
13         if x_j ∈ N_k then                                    /* neutral */
14             f_{c,j} ← {q_{c,j−1}, ∞}
15 return f;
```

Algorithm 3 gives pseudocode for the propagator. The intuition behind the algorithm is as follows. We emphasize again that 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 $S_{c,j-1}$ as the cost only depends on undetermined variables. 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 9 and 10 of the algorithm. Fig. 4a gives a schematic representation of these arguments.

If $x_j \in U_k$ then we have two options. We can obtain $S_{c,j}$ from $S_{c-1,j-1}$ by increasing $cst(S_{c-1,j-1})$ by one. This means that x_i will be covered by $last(S_{c,j})$. Alternatively, from $S_{c,j-1}$ by interrupting $last(S_{c,j-1})$. This is encoded in line 12 of the algorithm (Fig. 4b).

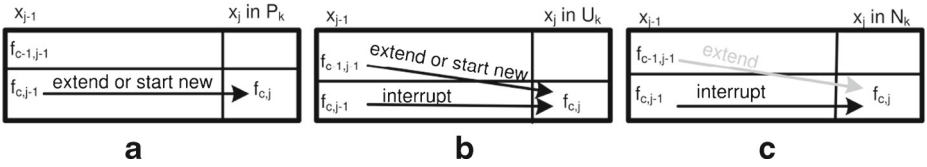

Fig. 4 Representation of one step of algorithm 3

If $x_j \in N_k$ then we do not increase the cost of $S_{c,j}$ compared to $S_{c,j-1}$. Moreover, we must interrupt $last(S_{c,j-1})$, line 14 (Fig. 4c, ignore the gray arc).

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 $S_{c',j'}$:

- $f_{c,j} < f_{c',j'}$ if $(q_{c,j} < q_{c',j'})$ or $(q_{c,j} = q_{c',j'}$ and $l_{c,j} < l_{c',j'})$.
- $f_{c,j} = f_{c',j'}$ if $(q_{c,j} = q_{c',j'}$ and $l_{c,j} = l_{c',j'})$.

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, then by the length of the last sequence in each set, $last(S_{c,j})$ and $last(S_{c',j'})$.

First, we prove two technical results.

Lemma 8 *Consider* SPRINGYFOCUS$([x_0, \ldots, x_{n-1}], y_c, len, k, z_c)$. *Let f be dynamic programming table returned by Algorithm 3. Then the non-dummy values of $f_{c,j}$ are consecutive in each column, so that there do not exist c, c', c'', $0 \le c < c' < c'' \le z_c^U$, such that $f_{c',j}$ is dummy and $f_{c,j}$, $f_{c'',j}$ are non-dummy.*

Proof We prove by induction on the length of the sequence. The base case is trivial as $f_{0,-1} = \{0, 0\}$ and $f_{c,-1} = \{\infty, \infty\}$, $c \in [-1] \cup [1, z_c^U]$. Suppose the statement holds for $j-1$ variables.

Suppose there exist c, c', c'', $0 \le c < c' < c'' \le z_c^U$, such that $f_{c',j}$ is dummy and $f_{c,j}$, $f_{c'',j}$ are non-dummy.

Case 1 Consider the case $x_j \in P_k$. By Algorithm 3, lines 9 and 10, $q_{c,j} \in [q_{c,j-1}, q_{c,j-1} + 1]$, $q_{c',j} \in [q_{c',j-1}, q_{c',j-1} + 1]$ and $q_{c'',j} \in [q_{c'',j-1}, q_{c'',j-1} + 1]$. As $f_{c',j}$ is dummy and $f_{c,j}$, $f_{c'',j}$ are non-dummy, $f_{c',j-1}$ must be dummy and $f_{c,j-1}$, $f_{c'',j-1}$ must be non-dummy. This violates induction hypothesis.

Case 2 Consider the case $x_j \in U_k$. By Algorithm 3, line 12, $q_{c,j} = \min(q_{c-1,j-1}, q_{c,j-1})$, $q_{c',j} = \min(q_{c'-1,j-1}, q_{c',j-1})$ and $q_{c'',j} = \min(q_{c''-1,j-1}, q_{c'',j-1})$. As $f_{c',j}$ is dummy, then both $f_{c'-1,j-1}$ and $f_{c',j-1}$ must be dummy. As $f_{c,j}$ is non-dummy, then one of $f_{c-1,j-1}$ and $f_{c,j-1}$ is non-dummy. As $f_{c'',j}$ is non-dummy, then one of $f_{c''-1,j-1}$ and $f_{c'',j-1}$ is non-dummy. We know that $c-1 < c \le c'-1 < c' \le c''-1 < c''$ or $c < c' < c''$. This leads to violation of induction hypothesis.

Case 3 Consider the case $x_j \in N_k$. By Algorithm 3, line 14, $q_{c,j} = q_{c,j-1}$, $q_{c',j} = q_{c',j-1}$ and $q_{c'',j} = q_{c'',j-1}$. Hence, $f_{c',j-1}$ is dummy and $f_{c,j-1}$, $f_{c'',j-1}$ are non-dummy. This leads to violation of induction hypothesis.

□

Proposition 4 *Consider* SPRINGYFOCUS$([x_0, \ldots, x_{n-1}], y_c, len, k, z_c)$. *Let* f *be dynamic programming table returned by Algorithm 3. The elements of the first row are non-dummy:* $f_{0,j}$, $j = -1, \ldots, n$ *are non-dummy.*

Proof We prove by induction on the length of the sequence. The base case is trivial as $f_{0,-1} = \{0, 0\}$. Suppose the statement holds for $j - 1$ variables.

Case 1 Consider the case $x_j \in P_k$. As $f_{0,j-1}$ is non-dummy then by Algorithm 3, lines 9–10, $f_{0,j}$ is non-dummy.

Case 2 Consider the case $x_j \in U_k$. Consider the condition $(l_{-1,j-1} \in [1, len) \wedge q_{-1,j-1} = q_{0,j-1}) \vee (q_{0,j-1} = \infty)$ at line 12. By the induction hypothesis, $q_{0,j-1} \neq \infty$. By the initialization procedure of the dummy row, $q_{-1,j-1} = \infty$. Hence, this condition does not hold and, by line 12, $f_{0,j}$ is non-dummy.

Case 3 Consider the case $x_j \in N_k$. As $f_{0,j-1}$ is non-dummy then by Algorithm 3, line 13, $f_{0,j}$ is non-dummy.

□

We can now prove an interesting monotonicity property of Algorithm 3.

Lemma 9 *Consider* SPRINGYFOCUS (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 that* $f_{c',j} \leq f_{c,j}$, $0 \leq c < c' \leq z_c^U$, $j = [0, \ldots, n-1]$.

Proof By transitivity and consecutivity of non-dummy values (Lemma 8) and the result that all elements in the 0th row are non-dummy (Proposition 4), it is sufficient to consider the case $c' = c + 1$.

We prove by induction on the length of the sequence. The base case is trivial as $f_{0,-1} = \{0, 0\}$ and $f_{c,0}$ are dummy, $c \in [0, z_c^U]$. Suppose the statement holds for $j - 1$ variables.

Consider the variable x_j. Suppose, by contradiction, that $f_{c,j} < f_{c+1,j}$. Then either $q_{c,j} < q_{c+1,j}$ or $q_{c,j} = q_{c+1,j}, l_{c,j} < l_{c+1,j}$. By induction hypothesis, we know that $f_{c,j-1} \geq f_{c+1,j-1}$, hence, either $q_{c,j-1} > q_{c+1,j-1}$ or $q_{c,j-1} = q_{c+1,j-1}, l_{c,j-1} \geq l_{c+1,j-1}$.

We consider three cases depending on whether x_j is a penalizing variable, an undetermined variable or a neutral variable.

Case 1 Consider the case $x_j \in P_k$. If $q_{c,j-1} = \infty$ then $q_{c+1,j-1} = \infty$ by the induction hypothesis. Hence, by Algorithm 3, line 9, $f_{c,j}$ and $f_{c+1,j}$ are dummy and equal. Suppose $q_{c,j-1} \neq \infty$. Then we consider four cases based on relative values of $q_{c,j'}, q_{c+1,j'}, l_{c,j'}, l_{c+1,j'}$, $j' \in \{j-1, j\}$.

- **Case 1a** Suppose $q_{c,j} < q_{c+1,j}$ and $q_{c,j-1} > q_{c+1,j-1}$. By Algorithm 3, lines 9 and 10, $q_{c,j} \geq q_{c,j-1}$ and $q_{c+1,j} \leq q_{c+1,j-1} + 1$. Hence, $q_{c,j} < q_{c+1,j}$ implies $\mathbf{q_{c+1,j-1}} < q_{c,j} < \mathbf{q_{c+1,j-1} + 1}$. We derive a contradiction.
- **Case 1b** Suppose $q_{c,j} < q_{c+1,j}$ and $q_{c,j-1} = q_{c+1,j-1}, l_{c,j-1} \geq l_{c+1,j-1}$. By Algorithm 3, lines 9 and 10, $q_{c,j} \geq q_{c,j-1}$ and $q_{c+1,j} \leq q_{c+1,j-1} + 1$. Hence, $q_{c,j} < q_{c+1,j}$ implies $\mathbf{q_{c+1,j-1}} = q_{c,j-1} \leq q_{c,j} < q_{c+1,j} \leq \mathbf{q_{c+1,j-1} + 1}$. Hence, $q_{c+1,j-1} = q_{c,j-1} = q_{c,j}$ and $q_{c+1,j} = q_{c+1,j-1} + 1$. As $q_{c,j-1} = q_{c,j}$ then $l_{c,j-1} \in [1, len)$ by

Algorithm 3 line 9. As $q_{c+1,j} = q_{c+1,j-1} + 1$ then $l_{c+1,j-1} \in \{len, \infty\}$ by Algorithm 3 line 10. This leads to a contradiction as $l_{c,j-1} \geq l_{c+1,j-1}$.

- **Case 1c** Suppose $q_{c,j} = q_{c+1,j}, l_{c,j} < l_{c+1,j}$ and $q_{c,j-1} > q_{c+1,j-1}$. Symmetric to Case 1b.
- **Case 1d** Suppose $q_{c,j} = q_{c+1,j}, l_{c,j} < l_{c+1,j}$ and $q_{c,j-1} = q_{c+1,j-1}, l_{c,j-1} \geq l_{c+1,j-1}$. By Algorithm 3, lines 9 and 10, $q_{c,j} \geq q_{c,j-1}$ and $q_{c+1,j} \leq q_{c+1,j-1} + 1$. Hence, $q_{c,j} = q_{c+1,j}$ implies $\mathbf{q_{c+1,j-1}} = q_{c,j-1} \leq q_{c,j} = q_{c+1,j} \leq \mathbf{q_{c+1,j-1}} + \mathbf{1}$. Therefore, either $q_{c,j} = q_{c,j-1} \wedge q_{c+1,j} = q_{c+1,j-1}$ or $q_{c,j} = q_{c,j-1} + 1 \wedge q_{c+1,j} = q_{c+1,j-1} + 1$.

  If $q_{c,j} = q_{c,j-1}$ and $q_{c+1,j} = q_{c+1,j-1}$ then $l_{c,j-1} \in [1, len)$ and $l_{c+1,j-1} \in [1, len)$ by Algorithm 3 line 9. Hence, $l_{c,j} = l_{c,j-1} + 1$ and $l_{c+1,j} = l_{c+1,j-1} + 1$. As $l_{c,j-1} \geq l_{c+1,j-1}$, then $l_{c,j} \geq l_{c+1,j}$. This leads to a contradiction with the assumption $l_{c,j} < l_{c+1,j}$.

  If $q_{c,j} = q_{c,j-1} + 1 \wedge q_{c+1,j} = q_{c+1,j-1} + 1$ then $l_{c,j-1} \in \{len, \infty\}$ and $l_{c+1,j-1} \in \{len, \infty\}$ by Algorithm 3 line 10. Hence, $l_{c,j} = 1$ and $l_{c+1,j} = 1$. This leads to a contradiction with the assumption $l_{c,j} < l_{c+1,j}$.

Case 2 Consider the case $x_j \in U_k$. If $q_{c,j-1} = \infty$ then $q_{c+1,j-1} = \infty$ by the induction hypothesis. Hence, by Algorithm 3, line 12, $f_{c,j}$ and $f_{c+1,j}$ are dummy and equal.

Suppose $q_{c,j-1} \neq \infty$. Then we consider four cases based on relative values of $q_{c,j'}, q_{c+1,j'}, l_{c,j'}, l_{c+1,j'}$, $j' \in \{j-1, j\}$.

- **Case 2a** Suppose $q_{c,j} < q_{c+1,j}$ and $q_{c,j-1} > q_{c+1,j-1}$. By Algorithm 3, line 12, we know that $q_{c+1,j-1} \leq q_{c+1,j} \leq q_{c,j-1}$ and $q_{c,j-1} \leq q_{c,j} \leq q_{c-1,j-1}$. By induction hypothesis, $q_{c+1,j-1} \leq q_{c,j-1} \leq q_{c-1,j-1}$. Hence, if $q_{c,j} \leq q_{c+1,j}$ then $\mathbf{q_{c,j-1}} \leq q_{c,j} \leq q_{c+1,j} \leq \mathbf{q_{c,j-1}}$. Therefore, if $q_{c,j} < q_{c+1,j}$ then we derive a contradiction.
- **Case 2b** Identical to Case 2b.
- **Case 2c** Suppose $q_{c,j} = q_{c+1,j}, l_{c,j} > l_{c+1,j}$ and $q_{c,j-1} > q_{c+1,j-1}$. As $q_{c,j-1} \neq q_{c+1,j-1}$ then $q_{c+1,j-1} = q_{c+1,j}$ (line 12). We also know $\mathbf{q_{c,j-1}} \leq q_{c,j} \leq q_{c+1,j} \leq \mathbf{q_{c,j-1}}$ from Case 1a. Putting everything together, we get $\mathbf{q_{c,j-1}} \leq q_{c,j} \leq q_{c+1,j-1} < \mathbf{q_{c,j-1}}$. This leads to a contradiction.
- **Case 2d** Suppose $q_{c,j} = q_{c+1,j}, l_{c,j} < l_{c+1,j}$ and $q_{c,j-1} = q_{c+1,j-1}, l_{c,j-1} \geq l_{c+1,j-1}$. As we know from Case 1a $q_{c+1,j-1} \leq q_{c+1,j} \leq q_{c,j-1}$, $q_{c,j-1} \leq q_{c,j} \leq q_{c-1,j-1}$ and $\mathbf{q_{c,j-1}} \leq q_{c,j} \leq q_{c+1,j} \leq \mathbf{q_{c,j-1}}$. Hence, $q_{c+1,j-1} = q_{c+1,j} = q_{c,j-1} = q_{c,j}$.

  Consider two subcases. Suppose $q_{c,j-1} < q_{c-1,j-1}$. Then $l_{c,j} = \infty$ (line 12). Hence, our assumption $l_{c,j} < l_{c+1,j}$ is false.

  Suppose $q_{c,j-1} = q_{c-1,j-1}$. If $l_{c-1,j-1} = len$ then $l_{c,j} = \infty$ (line 12). Hence, our assumption $l_{c,j} < l_{c+1,j}$ is false. Therefore, $l_{c-1,j-1} \in [1, len)$ and $l_{c,j-1} = l_{c-1,j-1} + 1$. By induction hypothesis as $q_{c+1,j-1} = q_{c,j-1} = q_{c-1,j-1}$ then $l_{c+1,j-1} \leq l_{c,j-1} \leq l_{c-1,j-1}$. Hence, $l_{c,j-1} \in [1, l_{c-1,j-1}] \subseteq [1, len)$. Therefore, $l_{c+1,j} = l_{c,j-1} + 1 \leq l_{c-1,j-1} + 1 = l_{c,j-1}$. This contradicts our assumption $l_{c,j} < l_{c+1,j}$.

Case 3 Consider the case $x_j \in N_k$. This case follows immediately from Algorithm 3, line 14, and the induction hypothesis.

□

Lemma 10 *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 1–4. The time complexity of Algorithm 3 is $O(n \max(z_c))$.

Proof We start by proving correctness of the algorithm. We use induction on the length of the sequence. Given $f_{c,j}$ we can reconstruct a corresponding set of sequences $S_{c,j}$ by traversing the table backward.

The base case is trivial as $x_1 \in P_k$, $f_{0,0} = \{1, 1\}$ and $f_{c,0} = \{\infty, \infty\}$. Suppose the statement holds for $j - 1$ variables.

Case 1 Consider the case $x_j \in P_k$. Note, that the cost can not be increased on seeing $x_j \in P_k$ as cost only depends on covered undetermined variables. By the induction hypothesis, $S_{c,j-1}$ satisfies conditions 1–4. The only way to obtain $S_{c,j}$ from $S_{c',j-1}$, $c' \in [0, z_c^U]$, is to extend $last(S_{c,j-1})$ to cover x_j or start a new sequence if $|last(S_{c,j-1})| = len$. If $S_{c,j-1}$ does not exist then $S_{c,j}$ does not exist. The algorithm performs this extension (lines 9 and 10). Hence, $S_{c,j}$ satisfies conditions 1–4.

Case 2 Consider the case $x_j \in U_k$. In this case, there exist two options to obtain $S_{c,j}$ from from $S_{c',j-1}$, $c' \in [0, z_c^U]$. The first option is to cover x_j. Hence, we need to extend $last(S_{c-1,j-1})$. Note that we should not start a new sequence if $last(S_{c-1,j-1}) = len$ as it is never optimal to start a sequence on seeing a neutral variable.

The second option is not to cover x_j. Hence, we need to interrupt $last(S_{c,j-1})$.

By Lemma 9 we know that $f_{c,j-1} \leq f_{c-1,j-1}$, $0 < c \leq C$. By the induction hypothesis, $S_{c,j-1}$ and $S_{c-1,j-1}$ satisfy conditions 1–4. Hence, $S_{c,j-1} \leq S_{c-1,j-1}$.

Consider two cases. Suppose $|S_{c,j-1}| < |S_{c-1,j-1}|$. In this case, it is optimal to interrupt $last(S_{c,j-1})$.

Suppose $|S_{c,j-1}| = |S_{c-1,j-1}|$ and $|last(S_{c,j-1})| \leq |last(S_{c-1,j-1})|$. If $|last(S_{c-1,j-1})| < len$ then it is optimal to extend $last(S_{c-1,j-1})$. If $|last(S_{c-1,j-1})| = len$ then it is optimal to interrupt $last(S_{c,j-1})$, otherwise we would have to start a new sequence to cover an undetermined variable x_j, which is never optimal. If $S_{c,j-1}$ and $S_{c-1,j-1}$ do not exist then $S_{c,j}$ does not exist. If $S_{c,j-1}$ does not exist then case analysis is similar to the analysis above.

This case-based analysis is exactly what Algorithm 3 does in line 12. Hence, $S_{c,j}$ satisfies conditions 1–4.

Case 3 Consider the case $x_j \in N_k$. Note that the cost can not be increased on seeing $x_j \in N_k$ as cost only depends on covered undetermined variables. By the induction hypothesis, $S_{c,j-1}$ satisfies conditions 1–4. The only way to obtain $S_{c,j}$ from $S_{c',j-1}$, $c' \in [0, z_c^U]$, is to interrupt $last(S_{c,j-1})$. If $S_{c,j-1}$ does not exist then $S_{c,j}$ does not exist. The algorithm performs this extension in line 14. Hence, $S_{c,j}$ satisfies conditions 1–4.

Regarding the worst case time complexity, it is clear that this algorithm requires $O(n \max(z_c)) = O(n^2)$ as we have $O(n \max(z_c))$ elements in the table and we only need to inspect a constant number of elements to compute $f(c, j)$. □

Example 8 Table 1 shows an execution of Algorithm 3 on WEIGHTEDFOCUS from Example 6. Note that $|P_0| = 5$. Hence, $z_c^U = \max(z_c) - |P_0| = 2$. As can be seen from the table, the constraint has a solution as there exists a set $S_{2,7} = \{s_{0,3}, s_{5,7}\}$ such that $|S_{2,7}| = 2$.

Table 1 An execution of Algorithm 3 on WEIGHTEDFOCUS from Example 6. Dummy values $f_{c,j}$ are removed

	$D(x_0)$	$D(x_1)$	$D(x_2)$	$D(x_3)$	$D(x_4)$	$D(x_5)$	$D(x_6)$	$D(x_7)$
c	[1, 1]	[0, 1]	[1, 1]	[1, 1]	[0, 1]	[1, 1]	[0, 1]	[1, 1]
0	{1, 1}	{1, ∞}	{2, 1}	{2, 2}	{2, ∞}	{3, 1}	{3, ∞}	{4, 1}
1		{1, 2}	{1, 3}	{1, 4}	{1, ∞}	{2, 1}	{2, ∞}	{3, 1}
$z_c^U = 2$					{1, 5}	{2, 1}	{2, 2}	{2, 3}

Bounds consistency To enforce BC on the sequence $[x_0, x_1, \ldots, x_{n-1}]$, we compute an additional DP table $b, b_{c,j}, c \in [0, z_c^U]$, $j \in [-1, n-1]$ on the reverse sequence of variables (i.e. $[x_{n-1}, \ldots, x_1, x_0]$).

Lemma 11 *Consider* WEIGHTEDFOCUS(X, y_c, len, k, z_c). *Bounds consistency can be enforced in* $O(n \max(z_c))$ *time.*

Proof 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 neighbour columns to the ith 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. Next, we provide a formal proof.

Consider dynamic programming tables f and b and a variable-value pair $x_i = v$. 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 $b_{c_2,n-i-2}$, $c_1, c_2 \in [0, z_c^U]$. We introduce two dummy variables x_{-1} and x_n, $D(x_{-1}) = D(x_n) = 0$ to keep uniform notations.

Consider a variable-value pair $x_i = v$, $v > k$. Note that it is sufficient to find a support one value v, $v > k$ as all values greater than k are indistinguishable. Due to Lemma 10 it is sufficient to consider only elements in the neighbouring columns to the ith column in f and b. Namely, the $(i-1)$th column in f and $(n-i-2)$ in b. The reason for that is that elements in these columns $f_{c_1,i-1}$ and $b_{c_2,n-i-2}$, $c_1, c_2 \in [0, z_c^U]$ correspond to sets of sequences, $S_{c_1,i-1}$ and $S_{c_2,n-i-2}$, that are optimal with respect to conditions 1–4 for the prefix $[x_0, \ldots, x_{j-1}]$ and the suffix $[x_{j+1}, \ldots, x_{n-1}]$, respectively. The main goal is to check whether we can 'glue' the corresponding partial covers $S_{c_1,i-1}$, $S_{c_2,n-i-2}$ with $x_i = v$ into a single cover S over all variables that satisfies the constraint. To glue $S_{c_1,i-1}$, $S_{c_2,n-i-2}$ and $x_i = v$ into a single cover we have few options:

- The first and the most expensive option is to create a new sequence s' of length 1 to cover x_i. Then the union $S = S_{c_1,i-1} \cup S_{c_2,n-i-2} \cup \{s'\}$ forms a cover s.t. $cst(S) = c_1 + c_2 + 1$ and $|S| = |S_{c_1,i-1}| + |S_{c_2,n-i-2}| + 1$.
- The second option is to extend $last(S_{c_1,i-1})$ to the right by one if $|last(S_{c_1,i-1})| < len$. Hence, the updated set $S'_{c_1,i-1}$ is identical to $S_{c_1,i-1}$ except the last sequence is increased by one element on the right. Then the union $S = S'_{c_1,i-1} \cup S_{c_2,n-i-2}$ forms a cover: $cst(S) = c_1 + c_2 + 1$ and $|S| = |S_{c_1,i-1}| + |S_{c_2,n-i-2}|$.
- The third option is to extend $last(S_{c_2,n-i-2})$ to the left by one if $|last(S_{c_2,n-i-2})| < len$. This case is symmetric to the previous case.

– The fourth and the cheapest option is to glue $last(S_{c_1,i-1})$, x_v and $last(S_{c_2,n-i-2})$ to a single sequence if $|last(S_{c_1,i-1})|+|last(S_{c_2,n-i-2})| < len$. Hence, $S'_{c_1,i-1} = S_{c_1,i-1} \setminus last(S_{c_1,i-1})$, $S'_{c_2,n-i-2} = S_{c_2,n-i-2} \setminus last(S_{c_2,n-i-2})$ and s' is a concatenation of $last(S_{c_1,i-1})$, $x = v$ and $last(S_{c_2,n-i-2})$]. Then the union $S = S'_{c_1,i-1} \cup S'_{c_2,n-i-2} \cup \{s'\}$ forms a cover: $cst(S) = c_1 + c_2 + 1$ and $|S| = |S_{c_1,i-1}| + |S_{c_2,n-i-2}| - 1$.

We can go over all pairs $f_{c_1,i-1}$ and $b_{c_2,n-i-2}$, $c_1, c_2 \in [0, z_c^U]$ and check the four cases above. If obtained cover S is such that $cst(S) \leq z_c^U$ and $|S| \leq \max(y_c)$ then we have found a support for $x_i = v$. Otherwise, $x_i = v$ does not have a support due to Lemma 10. However, if we need to consider all pairs $f_{c_1,i-1}$ and $b_{c_2,n-i-2}$, $c_1, c_2 \in [0, z_c^U]$ then finding a support takes $O((z_c^U)^2)$ time. We show next that it is sufficient to consider a linear number of pairs. We observe that in all four options above the cost of resulting cover S is $c_1 + c_2 + 1$. Therefore, we only need to consider pairs $f_{c_1,i-1}$ and $b_{c_2,n-i-2}$ such that $c_1 + c_2 + 1 \leq z_c^U$. Therefore, 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 $c_1 + c_2 + 1 \leq z_c^U$.

We prove by contradiction. Suppose, there exists a pair $f_{c_1,i-1}$ and $b_{c'_2,n-i-2}$ such that $c_1 + c'_2 + 1 \leq z_c^U$ and $S_{c_1,i-1}$ and $S_{c'_2,n-i-2}$ can be extended to a support. However, $S_{c_1,i-1}$ and $S_{c_2,n-i-2}$ can not be extended to a support for $x_i = v$, $c_1 + c_2 + 1 \leq z_c^U$ and $c'_2 < c_2$. By Lemma 9, we know $b_{c'_2,n-i-2} \leq b_{c_2,n-i-2}$. However, in this case, $|S_{c_1,i-1}| + |S_{c_2,n-i-2}| \leq |S_{c_1,i-1}| + |S_{c'_2,n-i-2}| \leq \max(y_c) + 1$. In the case of equality, we know that $last(S_{c_2,n-i-2}) < last(S_{c'_2,n-i-2})$. Hence, if $S_{c_1,i-1}$ and $S_{c'_2,n-i-2}$ can be extended to a support then $S_{c_1,i-1}$ and $S_{c_2,n-i-2}$ can be extended to a support. This leads to a contradiction.

Note that we do not need to search for each $f_{c_1,i-1}$ as we can find its pair $b_{c_2,n-i-2}$ in $O(1)$ due to consecutivity property of non-dummy values in each column (Lemma 8). Hence, we need $O(z_c^U) = O(\max(z_c))$ time to check for support for $x_i = v$.

Consider a variable-value pair $x_i = v$, $v \leq k$. Note that it is sufficient to find a support for one value v, $v \leq k$ as all values less than or equal to k are indistinguishable. We again consider all pairs in the neighbouring columns, $f_{c_1,i-1}$ and $b_{c_2,n-i-2}$ and consider how to 'glue' the corresponding partial covers $S_{c_1,i-1}$, $S_{c_2,n-i-2}$ with $x_i = v$ into a single cover S over all variables to satisfy the constraint. In this case, there is only one option to join $S_{c_1,i-1}$ and $S_{c_2,n-i-2}$. Then union $S = S_{c_1,i-1} \cup S_{c_2,n-i-2}$ forms a cover: $cst(S) = c_1 + c_2$ and $|S| = |S_{c_1,i-1}| + |S_{c_2,n-i-2}|$. We can go over all pairs $f_{c_1,i-1}$ and $b_{c_2,n-i-2}$, $c_1, c_2 \in [0, z_c^U]$ to check if such a pair exists. We again show that it is sufficient to consider a linear number of pairs. We observe that in all four options above the cost of resulting cover S is $c_1 + c_2$. Therefore, we only need to consider pairs $f_{c_1,i-1}$ and $b_{c_2,n-i-2}$ such that $c_1 + c_2 \leq z_c^U$. Therefore, 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 prove by contradiction. Suppose, there exists a pair $f_{c_1,i-1}$ and $b_{c'_2,n-i-2}$ such that $c_1 + c'_2 \leq z_c^U$ and $S_{c_1,i-1}$ and $S_{c'_2,n-i-2}$ can be extended to a support. However, $S_{c_1,i-1}$ and $S_{c_2,n-i-2}$ can not be extended to a support for $x_i = v$, $c_1 + c_2 \leq z_c^U$ and $c'_2 < c_2$. By Lemma 9, we know $b_{c'_2,n-i-2} \leq b_{c_2,n-i-2}$. However, in this case, $|S_{c_1,i-1}| + |S_{c_2,n-i-2}| \leq |S_{c_1,i-1}| + |S_{c'_2,n-i-2}| \leq \max(y_c)$. In the case of equality, we know that $last(S_{c_2,n-i-2}) < last(S_{c'_2,n-i-2})$. Hence, if $S_{c_1,i-1}$ and $S_{c'_2,n-i-2}$ can be extended to a support then $S_{c_1,i-1}$ and $S_{c_2,n-i-2}$ can be extended to a support. This leads to a contradiction.

Complexity We compute the tables f and b. Then we check for a support for two values v_1 and v_2, $v_1 \leq k$ and $v_2 > k$, in $D(x_i)$ in $O(\max(z_c))$ time for each variable x_i, $i = 0, \ldots, n-1$. Hence, the time complexity to enforce domain consistency is $O(n \max(z_c))$.

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 $c_1 + c_2 \leq z_c^U$. □

Example 9 Table 2 shows an execution of Algorithm 3 on the reversed sequence of variables x of FOCUS from Example 6.

Consider, for example, the variable x_4. To check if $x_4 = 1$ has as a support we need to consider two pairs: $f_{0,3}$, $b_{1,5}$ and $f_{1,3}$, $b_{0,5}$.

Consider the first pair: $f_{0,3} = \{2, 2\}$ and $b_{1,5} = \{1, 3\}$. As $|S_{0,3}| + |S_{1,5}| = 2 + 1 = \max(y_c) + 1 = 3$, we check whether we can merge $last(S_{0,3})$, $x_4 = 1$, and $last(S_{1,5})$. Hence, $|last(S_{0,3})| + |last(S_{1,5})| = 2 + 3 = len = 5$. Therefore, we cannot merge $last(S_{0,3})$, $x_i = 1$ and $last(S_{1,5})$ into a single sequence s' of length 5.

Consider the second pair: $f_{1,3} = \{1, 4\}$ and $b_{0,5} = \{2, 1\}$. As $|S_{1,3}| + |S_{0,5}| = 1 + 2 = \max(y_c) + 1 = 3$, $x_4 = 1$, we check whether we can merge $last(S_{1,3})$ and $last(S_{0,5})$. As $|last(S_{1,3})| + |last(S_{0,5})| = 4 + 1$ is equal to $len = 5$, we cannot merge $last(S_{1,3})$, $x_i = 1$ and $last(S_{0,5})$ into a single sequence s' of length at most 5. The second pair cannot be used to build a support for $x_4 = 1$. Hence, $x_4 = 1$ does not have a support.

To check if $x_4 = 0$ has as support we need to consider pairs: $f_{0,3}$, $b_{2,5}$ and $f_{1,3}$, $b_{1,5}$. Consider the first pair: $f_{0,3} = \{2, 2\}$ and $b_{2,5} = \{2, 1\}$. We have $|S_{0,3}| + |S_{2,5}| = 2 + 2 = \max(y_c) = 4$. Hence, $x_4 = 0$ has a support.

We observe a useful property of the constraint. If there exists $f_{c,n-1}$ such that $c < \max(z_c)$ and $q_{c,n-1} < \max(y_c)$ 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 $cst(S_X)$ and $|S_X|$ by at most one.

Decomposition with $O(n)$ variables and constraints Alternatively we can decompose WEIGHTEDFOCUS using $O(n)$ additional variables and constraints.

Table 2 An execution of Algorithm 3 on the reverse sequence of variables in WEIGHTEDFOCUS from Example 6. Dummy values $f_{c,j}$ are removed

	$D(x_0)$	$D(x_1)$	$D(x_2)$	$D(x_3)$	$D(x_4)$	$D(x_5)$	$D(x_6)$	$D(x_7)$
c	[1, 1]	[0, 1]	[1, 1]	[1, 1]	[0, 1]	[1, 1]	[0, 1]	[1, 1]
0	{4, 1}	{3, ∞}	{3, 2}	{3, 1}	{2, ∞}	{2, 1}	{1, ∞}	{1, 1}
1	{3, 1}	{2, ∞}	{2, 2}	{2, 1}	{1, ∞}	{1, 3}	{1, 2}	
$z_c^U = 2$	{2, 4}	{2, 3}	{2, 1}	{1, 5}	{1, 4}			

Given FOCUS(X, y_c, len, k), let z_c be a variable and $B = [b_0, b_1, \ldots, b_{n-1}]$ be a set of variables such that $\forall b_l \in B$, $D(b_l) = \{0, 1\}$. We can decompose WEIGHTEDFOCUS as follows:

WEIGHTEDFOCUS(X, y_c, len, k, z_c) $\Leftrightarrow$ FOCUS(X, y_c, len, k) $\wedge [\forall l, 0 \leq l < n, [(x_l \leq k) \wedge (b_l = 0)] \vee [(x_l > k) \wedge (b_l = 1)]] \wedge \sum_{l \in \{0,1,\ldots,n-1\}} b_l \leq z_c$.

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

Another interesting approach for solving WEIGHTEDFOCUS is to reformulate it as an integer linear program. If the constructed ILP is tractable as was the case for SPRINGYFOCUS, then we can obtain an alternative filtering algorithm for WEIGHTEDFOCUS. However, the approach that we used in Section 3.3 does not work for WEIGHTEDFOCUS. Recall that in Section 3.3 it was sufficient to consider $O(n)$ possible sequences with distinct starting points. It is essential that sequences have distinct starting points as this ensures that the resulting ILP has the consecutive ones property. By relaxing the disjointness requirement, we used these sequences to find a solution of SPRINGYFOCUSOVERLAP and transform it into a solution of SPRINGYFOCUS. The following example shows that the same approach does not work for WEIGHTEDFOCUS.

Example 10 Consider variables $X = [x_0, x_1, \ldots, x_5]$ with domains [1,{0,1},1,1, {0,1}, 1] and WEIGHTEDFOCUS (X, [2, 3], 3, 0, [0, 4]). Following approach in Section 3.3, we consider six sequences $S^o_X = \{s_{0,2}, s_{1,3}, s_{2,4}, s_{3,5}, s_{4,6}, s_{5,6}, s_{6,6}\}$. The cost of any solution that uses sequences from S^o_X is 6. However, there exists a solution of WEIGHTEDFOCUS with cost 4: $S_X = \{s_{0,1}, s_{2,3}, s_{5,5}\}$, $y_c = 3$ and $z_c = 4$.

5 Weighted springy FOCUS

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

5.1 Definition and filtering algorithm

Definition 10 Let y_c and z_c be two variables and k, len, h be three integers, such that $1 \leq len \leq |X|$ and $0 < h < len - 1$. An instantiation of $X \cup \{y_c\} \cup z_c$ satisfies WEIGHTEDSPRINGYFOCUS(X, y_c, len, h, k, z_c) iff 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. $\forall x_l \in X, x_l > k \Rightarrow \exists s_{i,j} \in S_X$ such that $l \in s_{i,j}$
3. $\forall s_{i,j} \in S_X, |\{l \in s_{i,j}, x_l \leq k\}| \leq h$
4. $\forall s_{i,j} \in S_X, j - i + 1 \leq len, x_i > k$ and $x_j > k$.
5. $\sum_{s_{i,j} \in S_X} |s_{i,j}| \leq z_c$.

We can again partition cost of S into two terms. $\sum_{s_{i,j} \in S} |s_{i,j}| = \sum_{s_{i,j} \in S} cst(s_{i,j}) + |P_k|$. However, $cst(s_{i,j})$ is the number of *undetermined* and *neutral* variables covered $s_{i,j}$, $cst(s_{i,j}) = |\{p | x_p \in U_k \cup N_k, x_p \in s_{i,j}\}|$ as we allow to cover up to h *neutral* variables.

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 $S^1_{c,j}$ and $S^2_{c,j}$ of minimum cardinality. The issue here is how to compare $last(S^1_{c,j})$ and $last(S^2_{c,j})$ if they cover a different number of neutral variables. Luckily, we can avoid this problem due to the following monotonicity property. If $last(S^1_{c,j})$ and $last(S^2_{c,j})$ are not equal to infinity then they both end at the same position j. Hence, if $last(S^1_{c,j}) \leq last(S^2_{c,j})$ then the number of neutral variables covered by $last(S^1_{c,j})$ is no larger than the number of neutral variables covered by $last(S^2_{c,j})$. Therefore, we can define order on sets $S_{c,j}$ as we did in Section 4 for WEIGHTEDFOCUS.

Our bounds disentailment detection algorithm for WEIGHTEDSPRINGYFOCUS mimics Algorithm 3. We show a pseudocode for it in Algorithm 4.

Algorithm 4: WEIGHTEDSPRINGYFOCUS($x_0, \ldots, x_{n-1}$)

```
1  for c ∈ −1..z_c^U do
2      for j ∈ −1..n − 1 do
3          f_{c,j} ← {∞, ∞, ∞};
4  f_{0,−1} ← {0, 0, 0} ;
5  for j ∈ 0..n − 1 do
6      for c ∈ 0..j do
7          if x_j ∈ P_k then                                   /* penalizing */
8              if (l_{c,j−1} ∈ [1, len)) ∨ (q_{c,j−1} = ∞);
9              then
10                 f_{c,j} ← {q_{c,j−1}, l_{c,j−1} + 1, h_{c,j−1}};
11             else
12                 f_{c,j} ← {q_{c,j−1} + 1, 1, 0};
13         if x_j ∈ U_k then                                   /* undetermined */
14             if (l_{c−1,j−1} ∈ [1, len) ∧ q_{c−1,j−1} = q_{c,j−1}) ∨ (q_{c,j−1} = ∞);
15             then
16                 f_{c,j} ← {q_{c−1,j−1}, l_{c−1,j−1} + 1, h_{c−1,j−1}};
17             else
18                 f_{c,j} ← {q_{c,j−1}, ∞, ∞} ;
19         if x_j ∈ N_k then                                   /* neutral */
20             if (l_{c−1,j−1} ∈ [1, len) ∧ h_{c−1,j−1} ∈ [1, h) ∧ q_{c−1,j−1} =
               q_{c,j−1}) ∨ (q_{c,j−1} = ∞);
21             then
22                 f_{c,j} ← {q_{c−1,j−1}, l_{c−1,j−1} + 1, h_{c−1,j−1} + 1};
23             else
24                 f_{c,j} ← {q_{c,j−1}, ∞, ∞} ;
25 return f;
```

We highlight two non-trivial differences between Algorithm 4 and Algorithm 3. The first difference is that each cell in the dynamic programming table $f_{c,j}$, $c \in [0, z^U_c]$, $j \in \{0, 1, \ldots, n-1\}$, where $z^U_c = \max(z_c) - |P_k|$, is a triple of values $q_{c,j}$, $l_{c,j}$ and $h_{c,j}$, $f_{c,j} = \{q_{c,j}, l_{c,j}, h_{c,j}\}$. The new parameter $h_{c,j}$ stores the number of neutral variables covered by $last(S_{c,j})$. The second difference is in the way we deal with neutral variables. If $x_j \in N_k$ then we have two options now. We can obtain $S_{c,j}$ from $S_{c-1,j-1}$ by increasing $cst(S_{c-1,j-1})$ by one and increasing the number of covered neutral variables by

$last(S_{c,j-1})$ (Fig. 4c, the gray arc). Alternatively, we can obtain $S_{c,j}$ from $S_{c,j-1}$ by interrupting $last(S_{c,j-1})$ (Fig. 4c, the black arc). BC can be enforced using two modifications of the corresponding algorithm for WEIGHTEDFOCUS.

Lemma 12 *Consider* WEIGHTEDSPRINGYFOCUS(X, y_c, len, h, k, z_c). *BC can be enforced in* $O(n \max(z_c))$ *time.*

Proof The main idea is identical to the proof of the WEIGHTEDFOCUS constraint. We only highlight the differences between the WEIGHTEDFOCUS constraint and the WEIGHTEDSPRINGYFOCUS constraint.

Consider a variable-value pair $x_i = v, v > k$. The only difference is in the fourth option. We denote $h(s_{i,j})$ the number of neutral variables covered by $s_{i,j}$. Similarly, $h(S) = \sum_{s_{i,j} \in S} h(s_{i,j})$.

– The fourth and the cheapest option is to glue $last(S_{c_1,i-1})$, x_v and $last(S_{c_2,n-i-2})$ to a single sequence if $|last(S_{c_1,i-1})| + |last(S_{c_2,n-i-2})| < len$ and $h(last(S_{c_1,i-1})) + h(last(S_{c_2,n-i-2})) \leq h$. Hence, $S'_{c_1,i-1} = S_{c_1,i-1} \setminus last(S_{c_1,i-1})$, $S'_{c_2,n-i-2} = S_{c_2,n-i-2} \setminus last(S_{c_2,n-i-2})$ and s' is a concatenation of $last(S_{c_1,i-1}), x = v$ and $last(S_{c_2,n-i-2})]$. Then the union $S = S'_{c_1,i-1} \cup S'_{c_2,n-i-2} \cup \{s'\}$ forms a cover: $cst(S) = c_1 + c_2 + 1$, $|S| = |S_{c_1,i-1}| + |S_{c_2,n-i-2}| - 1$ and $h(S) = h(last(S_{c_1,i-1})) + h(last(S_{c_2,n-i-2}))$.

The rest of the proof is analogous to WEIGHTEDFOCUS.

Consider a variable-value pair $x_i = v, v \leq k$. The main difference is that we have the second option to build a support. Namely, we glue $S_{c_1,i-1}$, x_i and $S_{c_2,n-i-2}$. Hence, if $c_1 + c_2 + 1 \leq z_c^U$, $|last(S_{c_1,i-1})| + |last(S_{c_2,n-i-2})| < len$ and $h(last(S_{c_1,i-1})) + h(last(S_{c_2,n-i-2})) < h$ then we can build a support for $x_i = v$. The rest of the proof is analogous to WEIGHTEDFOCUS. □

5.2 Decomposition

WEIGHTEDSPRINGYFOCUS can be encoded using the cost-REGULAR constraint [5]. Indeed, one can use two states T_0 and T_1 (in addition to the initial state) as follows. The state T_0 captures all values $v \leq k$ not included in any subsequence in S_X. The set of states T_1 captures the values belonging to a subsequence in S_X. The transition between T_0 and T_1 is quite straightforward following the semantic of WEIGHTEDSPRINGYFOCUS, however, the automaton is non-deterministic as on seeing $v \leq k$ in T_1, it either covers the variable or interrupts the last sequence. The automaton needs 3 counters to compute len, y_c and h. Hence, the time complexity of this encoding is $O(n^4)$. Unfortunately the non-deterministic cost-REGULAR is not implemented in any constraint solver to our knowledge. In fact REGULAR [7] and cost-REGULAR [5] are defined only with deterministic automatons. A possible way to deal with our non-deterministic situation is to transform it into a deterministic automaton. However this transformation is known to be exponential in the worst case. The worst case time complexity $O(n^4)$ is likely to get worse, however, domain consistency is guaranteed. In contrast, our algorithm takes just $O(n^2)$ time.

WEIGHTEDSPRINGYFOCUS can also be decomposed using the GCC constraint [14]. We define the following variables for all $i \in [0, \max(y_c) - 1]$ and $j \in [0, n-1]$: S_i the start of the ith sub-sequence. $D(S_i) = \{0, .., n + \max(y_c)\}$; E_i the end of the ith

sub-sequence. $D(E_i) = \{0, .., n + \max(y_c)\}$; T_j the index of the subsequence in S_X containing x_j. $D(T_j) = \{0, .., \max(y_c)\}$; Z_j the index of the subsequence in S_X containing x_j s.t. the value of x_j is less than or equal to k. $D(Z_j) = \{0, .., \max(y_c)\}$; $last_c$ the cardinality of S_X. $D(last_c) = \{0, .., \max(y_c)\}$; $Card$, a vector of $\max(y_c)$ variables having $\{0, .., h\}$ as domains.

WeightedSpringyFocus(X, y_c, len, h, k, z_c) $\Leftrightarrow$

$$
\begin{array}{ll}
(x_j \leq k) \vee Z_j = 0; & (x_j \leq k) \vee T_j > 0; \\
(x_j > k) \vee (T_j = Z_j); & (T_j \leq last_c); \\
(T_j \neq i) \vee (j \geq S_{i-1}); & (T_j \neq i) \vee (j \leq E_{i-1}); \\
(i > last_c) \vee (T_j = i) \vee & (j < S_{i-1}) \vee (j > E_{i-1}); \\
\forall q \in [1, \max(y_c) - 1] : & q \geq last_c \vee S_q > E_{q-1}; \\
\forall q \in [0, \max(y_c) - 1] : & q \geq last_c \vee E_q \geq S_q; \\
\forall q \in [0, \max(y_c) - 1] : & q \geq last_c \vee len > (E_q - S_q);
\end{array}
$$

$$
\begin{array}{l}
last_c \leq y_c; \qquad Gcc([T_0, .., T_{n-1}], \{0\}, [n - z_c]); \\
\qquad\qquad Gcc([Z_0, .., Z_{n-1}], \{1, .., \max(y_c)\}, Card);
\end{array}
$$

The main advantage of this decomposition is that it uses constraints that are available in most existing solvers. However, it hinders propagation, that is, *Bound Consistency* is no longer guaranteed. Consider the same example showing that WEIGHTEDFOCUS is stronger than the first decomposition using FOCUS. Let $n = 5$, $h = 0$, $k = 0$, $len = 3$, $D(x_0) = D(x_2) = \{1\}$, $D(x_3) = \{0\}$, $D(x_1) = D(x_4) = \{0, 1\}$, $D(y_c) = \{2\}$, and $D(z_c) = \{3\}$. Enforcing *Bound Consistency* using the above decomposition will keep the domain of x_4 equal to $\{0, 1\}$ whereas the value 1 has no support.

6 Experiments

6.1 Protocol

We use the Choco-2.1.5 solver on Intel Xeon E5-2640 processors (2.50GHz) under Linux. The source code as well as the reproduction steps are available at http://siala.github.io/focus/focus-details.pdf. We compare the propagators of our global constraints (denoted by F) of WEIGHTEDFOCUS and WEIGHTEDSPRINGYFOCUS against two decompositions with generic constraints (denoted by D_1 and D_2). For each benchmark, the comparison is performed using the same search strategies for the different constraint models. The first decomposition (D_1) is restricted to WEIGHTEDFOCUS and uses FOCUS as we explained in Section 4. Te second decomposition (D_2) is shown in Section 5.2 and uses constraints available in most CP solvers (such as GCC). We do not present experiments for the propagator of SPRINGYFOCUS because this propagator is linear in the number of variables and does not involve complex data structures, which leads to a behaviour similar to the case of FOCUS (see [12]). Although it makes an interesting connection between ILP and our framework, the ILP formulation of SPRINGYFOCUS cannot outperform this propagator.

We use the following presentation protocol for all tables. First, we give the number of solved instances ($\#sol$). Then, we report the CPU time *(Time)*, the number of nodes ($Nodes$), and the speed of exploration in terms of nodes explored per second ($Nodes/s$). In particular, we report the average ($avg.$) and the standard deviation ($dev.$) for these statistics

across all successful runs. The best results are shown with bold face fonts w.r.t. the number of solutions ($\#sol$).

6.2 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 half-season (in practice, the second half of the season is symmetric). 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 weeks 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. These requirements are expressed either using WEIGHTEDFOCUS or using its decomposition. The concentration of important matches into packages is obtained by minimizing y_c, while for each such value of y_c we obtain the global minimum length for packages by minimizing the sum of lengths.

Model In our model, inverse-channelling and ALLDIFFERENT constraints with the strongest propagation level express that each team plays once against each other team. With respect to the sport scheduling part (independently from the weights and WEIGHTEDFOCUS constraint or its decomposition), our model is inspired from Régin's paper on sport league scheduling [15], although some differences exist, in order to best fit with the available propagators of Choco-2.1.5. A pseudo-code of the model of the whole problem is provided in Fig. 5. We use the procedure GETCOLUMN(Integer[][] m, k) for extracting the k^{th} column of the matrix given as argument.

INPUT:
Int n; // number of teams, indexed from 0 to $n-1$
Int[][] wl; // size: $n \times n$ list of weights per possible couple of team
Int max(y_c), max(z_c) // WEIGHTEDFOCUS
MODEL:
IntVar[][] opponents; // size: $n \times (n-1)$, domain for each team: all other team numbers
IntVar[][] place; // size: $n \times (n-1)$, domain: {0,1} (away or home)
IntVar[][] breaks; // size: $n \times (n-2)$, domain: {0, 1} (no break, or break)
IntVar[] sum_breaks_by_team; // size: n
IntVar obj_number_of_breaks;
IntVar [][] match_weights; // size: $n \times (n-1)$, domain: {0,1}
IntVar [] sum_weights_by_day; // size: n-1
IntVar y_c, z_c; // WEIGHTEDFOCUS
int *len*, k; // WEIGHTEDFOCUS
$\forall i \in 0..n-1$, ALLDIFFERENT(opponents[i]);
$\forall k \in 0..n-2$, ALLDIFFERENT(*getColumn*(opponents, k));
$\forall i \in 0..n-1, \forall k \in 0..n-2$, opponents[$i$][$k$] = $j \Leftrightarrow$ opponents[j][k] = i;
$\forall i \in 0..n-1, \forall k \in 0..n-2$, places[$i$][$k$] = 0 $\wedge$ opponents[i][k] = $j \Leftrightarrow$ place[j][k] = 1;
$\forall i \in 0..n-1, \forall k \in 0..n-2$, places[$i$][$k$] = 1 $\wedge$ opponents[i][k] = $j \Leftrightarrow$ place[j][k] = 0;
$\forall i \in 0..n-1, \forall j \in 0..n-3$, breaks[$i$][$j$] = (place[$i$][$j$]=place[$i$][$j+1$]); // reification
$\forall i \in 0..n-1$, sum_breaks_by_team[i] = sum of breaks of each team;
obj_number_of_breaks = $\sum_{i \in 0..n-1}$ sum_breaks_by_team[i];
$\forall i \in 0..n-2$ sum_weights_by_day[i] = sum of weights of each day;
$\forall i \in 0..n-1, \forall k \in 0..n-2$, opponents[$i$][$k$] = $j \Leftrightarrow$ match_weights[i][k] = wl[i][j];
WEIGHTEDFOCUS(sum_weights_by_day, y_c, *len*, k, z_c);

Fig. 5 Model of the SLS benchmark

Search strategy We use the following search strategy: assign first the sum of breaks by team, then the breaks and then the places, using for each group the *DomOverWDeg* variable selection strategy with the lowest values assigned first [2]. We fix the matches of the first team and then minimize z_c while the number of breaks is at its theoretical minimum $(n-2)$ and we arbitrary fix the maximum value of y_c.

In our context, using *DomOverWDeg* does not affect the comparison between the decomposition and the global constraint approach. Using a static search strategy leads to poor results concerning the sport league scheduling part of the problem, but this part is common to the decomposition and the global constraint models. Regarding TV broadcast packages, the results with WEIGHTEDFOCUS are almost the same with *DomOverWDeg* and if we use a static search strategy for the variables expressing weights and sum of weights. Using the decomposition approach, the results are better with *DomOverWDeg*. We present the results obtained for each model using *DomOverWDeg* in Table 3 and using a static branching (lexicographic exploration with minimum value) in Table 4 .

We consider the results with 16, 18, and 20 teams, on sets of 50 instances with 10 random important games and a limit of 400K backtracks. $\max(y_c) = 3$ and we search for one solution with $h \leq 7$ (instances n-1), $h \leq 6$ (n-2) and $h \leq 5$ (n-3). Note that the models with 18 and 20 teams are not shown in Table 4 because no solution was found with the static branching.

Table 3 shows clearly that the model using the global propagator dominates the decomposition on this problem. The difference of resolved instances between the two models increases with the instance size. For example with instances 20_3 the filtering algorithm solves 39 instances out of 50 whereas the decomposition solves only 29 of the instances. The new filtering does not require additional amount of time, and in fact it is faster than the average CPU time of the decomposition in general.[1]

There are many cases where the shape of the search tree differs between the two methods in terms of nodes. For instance, with 18_1, enforcing domain consistency deplores 1876 nodes whereas the decomposition explores at least three times this number (i.e. 6040). The extra filtering of the global constraint does help a lot by pruning more unsatisfiable subtrees which guides the heuristic towards solutions. It should be noted, however, that the decomposition explores faster the search tree. This behaviour is expected as decomposition leads to simpler filtering that is likely to be faster in general. It should be noted also that the standard deviation in almost all the cases was smaller with the complete filtering.

Regarding the results with the static branching, one can confirm that the models behave poorly as expected (Table 4). However, the performances trend is the same. More importantly, the results of the complete filtering are more robust than the decomposition. Take for instance the results of 16_2. The standard deviation of the nodes is 37 using the global constraint and 1749 using the decomposition.

6.3 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.

[1] Recall that the average CPU includes only the runtime of the successful runs.

Table 3 SLS with WEIGHTEDFOCUS and its decomposition using *DomOverWDeg*

	16_1							16_2							16_3						
	#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s	
		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg .	dev.	avg.	dev.
F	**50**	0.4	1.1	555	1450	1437	187	**50**	0.03	3.1	2271	4163	1339	159	**47**	7.4	16	9679	21555	1270	124
D_1	**50**	0.9	3.7	2549	12488	1578	434	49	2.8	6.9	4965	13118	1455	347	44	8.5	16.5	12702	25728	1300	186
	18_1							18_2							18_3						
	#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s	
		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg .	dev.	avg.	dev.
F	**49**	2	7	1876	6460	1050	157	**49**	7.8	16.3	8921	23642	1026	176	**42**	12	16	12062	17158	946	108
D_1	**49**	3.6	9.5	6040	16230	1290	432	46	9.7	16.3	15063	31795	1207	412	37	11.5	14.7	12648	16097	1022	153
	20_1							20_2							20_3						
	#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s	
		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg .	dev.	avg.	dev.
F	**49**	7.5	15.2	6265	12237	866	174	**45**	15.5	21.1	13576	20727	827	121	**39**	23.7	28.2	19274	22659	828	98
D_1	43	14.1	35	19879	54977	1013	318	35	13.2	18393	17318	28917	1017	277	29	18.2	22.7	16373	20769	861	126

Table 4 SLS with WEIGHTEDFOCUS and its decomposition using a static branching

	16_1							16_2							16_3						
	#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s	
		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg.	dev.	avg.	dev.
F	**23**	0	0	6553	39	7144	1353	**15**	0	0	3911	37	6346	750	**7**	0	0	1673	31	5741	983
D_1	21	0	0	6505	51	7115	1885	13	0.3	1	10538	1749	6423	2191	**7**	0.5	1.1	7168	1917	5043	1660

INPUT:
Int n ; // size of the sequence
Int m; // number of among constraints
Int[] s, e, l, u; // four vectors of m integers used for the among constraints.
Int len, h ; // used for WEIGHTEDSPRINGYFOCUS
MODEL:
IntVar[] X; // size: n , domain $\{0, 1\}$
IntVar y_c, z_c ; // used for WEIGHTEDSPRINGYFOCUS
$\forall d \in 1..m,\ l[d] \leq \sum_{i=s[d]}^{i=e[d]} X[i] \leq u[d]$; // the set of AMONG constraints
WEIGHTEDSPRINGYFOCUS(X, y_c , len , h , 0, z_c);

Fig. 6 Model of the cumulative scheduling with rentals problem

Model The problem is stated in a very simple way by bucketing time with {0,1} variables indicating whether a machine is rented or not for covering this time point. We define a conjunction of one WEIGHTEDSPRINGYFOCUS($X, y_c, len, h, 0, z_c$) with a set of AMONG constraints. The decision version of the problem is presented in Fig. 6. 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. Therefore, we build a Pareto frontier over two cost variables, as Fig. 7 shows for one of the instances of this problem. More specifically, we start by minimizing y_c, then immediately try to minimize z_c while fixing y_c to its minimum. Afterwards, we repeatedly increment y_c by 1 then try to find the correspondent minimal value of z_c. The process stops when either a maximum number of iterations is reached or no improvement on z_c is obtained.

Search strategy We use again two different search strategies: *DomOverWDeg* and static lexicographical exploration; both with the lowest values assigned first.

Figure 7 confirms the gain of flexibility illustrated by Fig. 1 in Section 3: allowing $h = 1$ variable with a low cost value into each sequence leads to new solutions, with significantly lower values for the target variable y_c.

We generated instances having a fixed length of sub-sequences of size 20 (i.e., $len = 20$), 50 % as a probability of posting an *Among* constraint for each (i, j) s.t. $j \geq i + 5$ in the

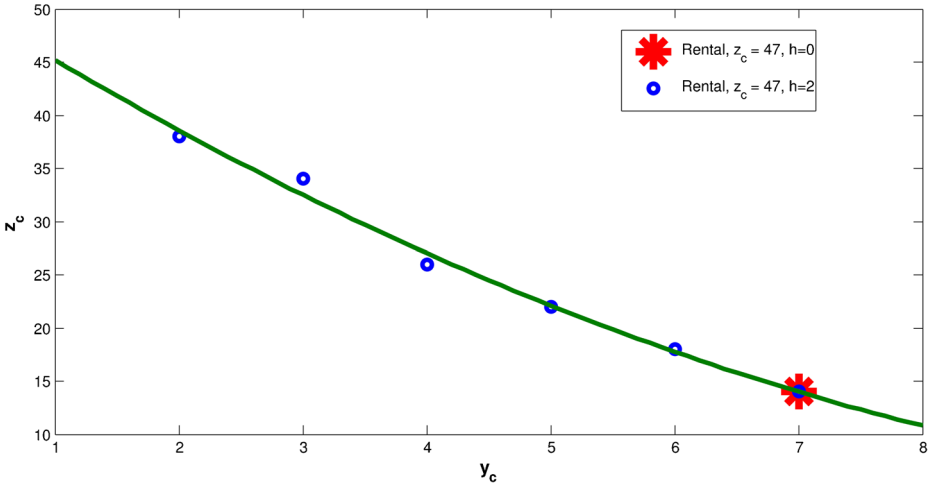

Fig. 7 Pareto frontier for scheduling with rentals

Table 5 Scheduling with rentals using DomOverWDeg

	40							43						
	#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s	
		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg.	dev.	avg.	dev.
h=0														
F	**20**	40	122	135018	423579	1041	869	**20**	90	269	259805	800815	872	574
D_1	**20**	59.09	210	212629	769672	1092	847	**20**	119	391	372541	1271169	937	665
h=1														
F	**20**	95	176	341394	640856	2844	2300	**20**	252	689	801909	2235387	2556	1769
h=2														
F	**20**	96	179	341134	631228	2792	2354	**20**	257	665	815084	2191171	2560	1791
	45							47						
	#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s	
		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg.	dev.	avg.	dev.
h=0														
F	**20**	212	960	565173	2507057	853	701	**20**	326	547	779999	1300376	716	223
D_1	**20**	295	1405	840967	4006454	900	723	**20**	419	740	1038147	1876769	748	309
h=1														
F	**20**	568	1455	1642111	3989712	2613	2598	19	1070	1744	2862135	4794331	2568	2180
h=2														
F	**20**	594	1380	1696763	3797544	2588	2467	19	1119	1767	2978121	4621013	2561	2125
	50													
	#*sol*	Time		Nodes		Nodes/s								
		avg.	dev.	avg.	dev.	avg.	dev.							
h=0														
F	**17**	645	1444	1383185	3029220	676	329							
D_1	14	649	947	1425712	2075470	691	295							
h=1														
F	**11**	1534	1362	3774669	3020671	2448	1401							
h=2														
F	**11**	1618	1964	3953820	4409741	2431	1382							

Table 6 Scheduling with rentals using a static branching

	40							43						
	#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s	
		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg.	dev.	avg.	dev.
h=0														
F	**20**	13	27	60378	139508	1239	1147	**20**	25	60	98351	262913	1089	871
D_1	**20**	16	31	80002	169747	1393	1210	**20**	30	75	133450	344122	1223	902
h=1														
F	**20**	83	118	349495	522644	3096	2595	**20**	284	1021	1102987	4530456	2787	1803
h=2														
F	**20**	83	118	349488	522638	3103	2575	**20**	285	1038	1102980	4530356	2791	1849

	45							47						
	#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s	
		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg.	dev.	avg.	dev.
h=0														
F	**20**	68	220	260331	882753	1094	930	**20**	91	213	309481	783447	932	396
D_1	**20**	85	276	352155	1149913	1235	965	**20**	110	264	428014	1110529	1078	501
h=1														
F	**18**	1037	631	3723494	2163719	2976	749	**2**	1205	202	3935339	865505	2560	743
h=2														
F	**18**	1041	638	3723480	2163748	2977	772	**2**	1207	203	3935354	865522	2584	760

	50						
	#*sol*	Time		Nodes		Nodes/s	
		avg.	dev.	avg.	dev.	avg.	dev.
h=0							
F	**20**	216	563	650832	1775355	893	728
D_1	**20**	260	688	895516	2461464	1035	811

sequence. Each set of instances corresponds to a unique sequence size ({40, 43, 45, 47, 50}) with 20 different seeds.

We summarize these tests in Tables 5 and 6. Results with decomposition are very poor. We therefore do not show them in these tables.

The performances in this problem with *DomOverWDeg* are very similar to the sports league scheduling problem. The global filtering completely outperforms the decomposition with GCC as we said. Regarding the first decomposition (D_1), it behaves relatively well on the first four sets 40, 43, 45, 47 and slightly worse than the global constraint in the set 50 (i.e. only 14 solved instances compared to 17 instances with F).

Using the static branching on this particular problem was very beneficial. There is no significant performance differences betweens the two models F and D_1. Indeed, they find the same number of solution in all instances with $h = 0$. The average runtime is slightly but constantly better with the global filtering. The number of nodes is also smaller. However, overall, there was no significant difference between the two models.

It should be noted that in both branching strategies, the standard deviation is better with the global constraint than the decomposition.

6.4 Sorting chords

We need to sort n distinct chords. Each chord is a set of at most p notes played simultaneously. The goal is to find an ordering that minimizes the number of notes changing between two consecutive chords (Fig. 8).

Model The full description and a CP model is in [12]. Figure 6 provides a pseudo-code for this problem. The main difference here is that instead of minimizing either z_c or y_c, we build a Pareto frontier over these two cost variables (the same way performed with the previous benchmark), using WEIGHTEDSPRINGYFOCUS and its decompositions. We generated 4 sets of instances distinguished by the numbers of chords ({14, 16, 18, 20}). We fixed the length of the subsequences and the maximum notes for all the sets then changed the seed for each instance.

Search strategy As in the Sports League Scheduling benchmark, we present the results obtained for each model, i.e., the model that uses WEIGHTEDSPRINGYFOCUS and the models with its decompositions. The search strategy is *DomOverWDeg* with the lowest values assigned first (Table 7). The static branching performs very poorly on these instances and is therefore not shown here.

INPUT:
Int n; // number chords, indexed from 0 to $n-1$
Int[][] *costMatrix;* // size: $n \times n$, matrix of costs between pairs of chords
Int *len* , h, k; // WEIGHTEDSPRINGYFOCUS
MODEL:
IntVar[] Chords; // size: n, domain $\{0, 1, \ldots, n-1\}$
IntVar[] Costs; // size: $n-1$, domain: all possible costs
Int nChange; // threshold from which a cost is considered as high
IntVar y_c, z_c; // WEIGHTEDSPRINGYFOCUS
$\forall i \in 0..n-2$, TABLE(Chords[i], Chords[$i+1$], Costs[i]); // cost of each pair
ALLDIFFERENT(Chords);
WEIGHTEDSPRINGYFOCUS(Costs, y_c, *len* , h, k, z_c);

Fig. 8 Model of the sorting chords benchmark

Table 7 Sorting chords

	14							16						
	#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s	
		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg.	dev.	avg.	dev.
h=0														
F	**30**	2	7	17577	73168	2923	5350	29	4	13	29963	109164	2737	3655
D_1	**30**	2	6	15072	55030	2920	4841	**30**	5	14	35702	117407	2968	3604
D_2	**30**	27	127	154422	739803	2437	3681	20	24	97	125863	522175	2380	2222
h=1														
F	**30**	2	12	16224	112110	3707	12269	**30**	75	980	698888	10086454	4430	13202
D_2	**30**	29	135	144228	682095	2252	3268	20	28	112	125375	526964	2107	2029
h=2														
F	**30**	2	12	17237	112323	3649	12000	**29**	31	243	249125	2121349	3883	10085
D_2	**30**	31	146	157560	752363	2223	3230	20	29	112	134773	549205	2163	1964
	18							20						
	#*sol*	Time		Nodes		Nodes/s		#*sol*	Time		Nodes		Nodes/s	
		avg.	dev.	avg.	dev.	avg.	dev.		avg.	dev.	avg.	dev.	avg.	dev.
h=0														
F	24	23	80	117113	417085	2022	1864	**10**	639	2233	3415012	12383754	2849	2442
D_1	**25**	59	206	337787	1213605	2283	2406	9	666	1727	3534231	9344656	3023	3214
D_2	**9**	235	837	926145	3357408	1660	1594	3	252	572	844450	1877702	2202	982
h=1														
F	**24**	397	2345	2500105	15264782	4123	9485	**9**	444	1109	2257348	6036524	3023	3215
D_2	9	263	931	947522	3425814	1527	1199	3	284	674	859046	2017445	1932	855
h=2														
F	**25**	336	1709	2122557	11238548	4468	11413	**11**	607	1719	3187854	9777345	2937	3120
D_2	9	223	703	804368	2637317	1482	1248	4	384	728	1091963	2074681	1805	1097

The main observation from Table 7 is that when $h = 0$, the first decomposition D_1 performs as good as the complete filtering in general. With 16 and 18 chords, D_1 finds an additional solution compared to the complete filtering F. The average nodes, and the average nodes explored per second are very similar in both models. The standard deviation is also very similar with all statics in general.

The decomposition using GCC performs much better than the previous problem but it is outperformed by WEIGHTEDSPRINGYFOCUS. For example, on instances with $h = 2$ using 18 chords, it finds 9 solutions whereas the complete filtering finds 25.

7 Conclusion

We have presented flexible tools for capturing the concept of concentrating costs. Our contribution highlights the expressive power of constraint programming, in comparison with other paradigms where such a concept would be very difficult to represent. We have shown a connection between our constraint and ILP. Our experiments have demonstrated the effectiveness of the proposed new filtering algorithms.

References

1. Ahuja, R.K., Magnanti, T. L., & Orlin, J.B. (1993). Network flows: Theory, algorithms, and applications.
2. Boussemart, F., Hemery, F., Lecoutre, C., & Sais, L. (2004). Boosting systematic search by weighting constraints. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)* (pp. 482–486).
3. Dasgupta, S., Papadimitriou, C.H., & Vazirani, U.V. (2006). *Algorithms*. McGraw-Hill.
4. De Clercq, A., Petit, T., Beldiceanu, N., & Jussien, N. (2011). Filtering algorithms for discrete cumulative problems with overloads of resource. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP'11)* (pp. 240–255).
5. Demassey, S., Pesant, G., & Rousseau, L.-M. (2006). A cost-regular based hybrid column generation approach. *Constraints*, *11*(4), 315–333.
6. Maher, M., Narodytska, N., Quimper, C.-G., & Walsh, T. (2008). Flow-based propagators for the sequence and related global constraints. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming (CP'08)* (pp. 159–174).
7. Pesant, G. (2004). A regular language membership constraint for finite sequences of variables. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP'04)* (pp. 482–495).
8. Pesant, G. (2001). A filtering algorithm for the stretch constraint. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP'01)* (pp. 183–195).
9. Pesant, G., & Spread, J.-C.Régin. (2005). A balancing constraint based on statistics. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)* (pp. 460–474).
10. Petit, T., & Poder, E. (2008). Global propagation of practicability constraints. In *Proc. CPAIOR, volume 5015* (pp. 361–366).
11. Petit, T., & Régin, J.-C. (2011). The ordered distribute constraint. *International Journal on Artificial Intelligence Tools*, *20*(4), 617–637.
12. Petit, T. (2012). Focus: A constraint for concentrating high costs. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP'12)*.
13. Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th National Conference on Artificial intelligence (AAAI'94)* (Vol. 1, pp. 362–367). American Association for Artificial Intelligence.
14. Régin, J.-C. (1996). Generalized arc consistency for global cardinality constraint. In *Proceedings of the 14th National Conference on Artificial intelligence (AAAI'98)* (pp. 209–215).

15. Régin, J.-C. (2001). Minimization of the number of breaks in sports scheduling problems using constraint programming. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, *57*, 115–130.
16. Rossi, F., van Beek, P., & Walsh, T. (2006). *Handbook of constraint programming (Foundations of Artificial Intelligence)*. New York: Elsevier Science Inc.
17. Schaus, P., Deville, Y., Dupont, P., & Régin, J.-C. (2007). The deviation constraint. In *Proc. CPAIOR* (Vol. 4510, pp. 260–274).
18. Schaus, P., Van Hentenryck, P., & Régin, J.-C. (2009). Scalable load balancing in nurse to patient assignment problems. In *Proc. CPAIOR, volume 5547 of Lecture Notes in Computer Science* (pp. 248–262). Springer.
19. Wagner, A., & Harvey, M. (1962). Optimal capacity scheduling I. *Operations Research*, *10*(4), 518–532.