Three generalizations of the FOCUS constraint
Nina Narodytska, Thierry Petit, Mohamed Siala, Toby Walsh

To cite this version:
Three Generalizations of the FOCUS Constraint

Nina Narodytska · Thierry Petit · Mohamed Siala · Toby Walsh

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.

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 rich paradigm to express and filter such constraints. In particular, several constraints have been proposed for obtaining well-balanced solutions [9,17,11]. 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...
The standard capacity constraints are exceeded in the new activity. Suppose that an additional activity has to be scheduled on this resource. The new activity has a duration of 3 days, each of which consumes 2 units of capacity. The following sequence (denoted \( S_1 \)) shows the new resource consumption if we start the new activity at day 1:

\[
\begin{array}{cccccccccc}
\text{x_0} & \text{x_1} & \text{x_2} & \text{x_3} & \text{x_4} & \text{x_5} & \text{x_6} & \text{x_7} & \text{x_8} & \text{x_9} \\
4 & 2 & 4 & 2 & 2 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

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 \( x_2 \).

Unfortunately, the usefulness of \( \text{FOCUS} \) is hindered by the rigidity of its semantics. For example, we might be able to rent a machine from Monday to Saturday but not use it on Sunday. It is a pity to miss such a solution with a smaller number of rental intervals. For example, we might be able to rent a machine from Monday to Sunday but concentrate (non null) excesses in a small number of intervals, each of length at most 2 days. \( \text{FOCUS} \) can be set to terminate when all the strictly positive variables in \( X \) with only 2 sequences of length \( \leq 2 \).

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

Example 1 Let \( k = 0 \), \( D(y_c) = \{2\} \), \( X = \{x_0, \ldots, x_9\} \), \( D(x_0) = \{1\} \), \( D(x_1) = \{3\} \), \( D(x_2) = \{1\} \), \( D(x_3) = \{0\} \), \( D(x_4) = \{1\} \), \( D(x_5) = \{0\} \). If \( \text{len} = 6 \), then \( \text{FOCUS}(X, y_c, \text{len}, k) \) is satisfied since we can have 2 disjoint sequences of length \( \leq 6 \) of consecutive variables with a value strictly positive, i.e., \( \langle x_0, x_1, x_2 \rangle \), and \( \langle x_4 \rangle \). If \( \text{len} = 2 \), \( \text{FOCUS}(X, y_c, \text{len}, k) \) becomes violated since it is impossible to include all the strictly positive variables in \( X \) with only 2 sequences of length \( \leq 2 \).

\( \text{FOCUS} \) 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. \( \text{FOCUS} \) can be set to concentrate (non null) excesses in a small number of intervals, each of length at most \( \text{len} \).

Unfortunately, the usefulness of \( \text{FOCUS} \) 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 \( \text{FOCUS} \) 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, \( \text{FOCUS} \) only considers the number of disjoint sequences, and does not consider their length.
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., \( \langle x_0, x_1, x_3, x_4 \rangle \)). 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:

\[
[x_0, \ldots, x_9] \quad 4 \quad 2 \quad 4 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2
\]

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. WEIGHTEDFOCUS adds a variable to count the length of sequences, equal to the number of variables taking a value \( v > k \). The most generic one, WEIGHTEDSPRINGYFOCUS, combines the semantics of SPRINGYFOCUS and WEIGHTEDFOCUS. 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 \( \text{Scope}(C) \) 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 $\text{Scope}(C)$ from its domain which satisfies $C$. A bounds support for a constraint $C$ is a tuple that assigns a value to each variable in $\text{Scope}(C)$ 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 $x_i \in \text{Scope}(C)$, 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 $x_i \in \text{Scope}(C)$, 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 Figure 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($X, ye = 1, 5, 3$) is imposed then this activity must start at day 1 (solution B). We have one 5 day rental interval.
Three Generalizations of the FOCUS Constraint

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 and 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.

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. $\forall i \in X, x_i > k \Rightarrow \exists s_{i,j} \in S_X$ 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 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 \( s \subseteq X \), denoted \( fc(s) \), is defined as follows:

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

**Definition 4** Given \( x_l \in X \), we consider three quantities.

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

**Proof** By construction from Definitions 2 and 4. \( \square \)

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

**Definition 5** \( plen(x_l) \) 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 \( fc([x_0, x_1, \ldots, x_l]) = 0 \) if \( \forall s_{i,j} \in S_{[x_0, x_1, \ldots, x_l]}, j \neq l \).

**Definition 6** \( \text{card}(x_l) \) 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 \( \forall s_{i,j} \in S_{[x_0, x_1, \ldots, x_l]}, j \neq l \). \( \text{card}(x_l) \) assumes that \( x_l \geq k \) and has to be decreased 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) \( p(x_0, v_\leq) = 0 \) if \( \min(x_0) \leq k \), and \( n + 1 \) otherwise; \( p_S(x_0, v_\leq) = n + 1 \); \( p(x_0, v_>) = 1 \) if \( \max(x_0) > k \) and \( n + 1 \) otherwise; \( plen(x_0) = 1 \) if \( \max(x_0) > k \) and \( 0 \) otherwise; \( \text{card}(x_0) = 0 \).

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

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

**Lemma 2** (\( p(x_l, v_\leq) \)) If \( \min(x_l) \leq k \) then \( p(x_l, v_\leq) = \min(p(x_{l-1}, v_\leq), p(x_{l-1}, v_>) ) \), else \( p(x_l, v_\leq) = n + 1 \).
We thus have for each quantity two values for each variable \( p \). By construction from Definition 5 and Lemmas 1, 2, 3 and 4.

Proposition 1 \((x_{i-1}, v_{\leq})\) If \( \min(x_i) \leq k \) then \( p_{\geq}(x_{i-1}, v_{\leq}) \) must not be considered: it would imply that a sequence in \( S_X \) ends by a value \( v \leq k \) for \( x_{i-1} \). From Property 1, the focus cardinality of the previous sequence is \( \min(p(x_{i-1}, v_{\geq}), p(x_{i-1}, v_{\leq})) \). □

Lemma 3 \((p_{\leq}(x_i, v_{\leq}))\) If \( \min(x_i) > k \), \( p_{\leq}(x_i, v_{\leq}) = n + 1 \).
Otherwise, if \( \text{plen}(x_{i-1}) \in \{0, \text{len} - 1, \text{len}\} \) or \( \text{card}(x_{i-1}) = h \) then \( p_{\leq}(x_i, v_{\leq}) = n + 1 \), else \( p_{\leq}(x_i, v_{\leq}) = \min(p_{\leq}(x_{i-1}, v_{\leq}), p(x_{i-1}, v_{\geq})). \)

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

Lemma 4 \((p(x_i, v_{\geq}))\) If \( \max(x_i) \leq k \) then \( p(x_i, v_{\geq}) = n + 1 \).
Otherwise, \( p_{\leq}(x_{i-1}) \in \{0, \text{len}\} \) \( p(x_i, v_{\geq}) = \min(p(x_{i-1}, v_{\geq}) + 1, p(x_{i-1}, v_{\leq}) + 1) \), else \( p(x_i, v_{\geq}) = \min(p(x_{i-1}, v_{\geq}) + 1, p(x_{i-1}, v_{\leq}) + 1) \).

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

Proposition 1 \((\text{plen}(x_i))\) If \( \min(p_{\leq}(x_{i-1}, v_{\leq}), p(x_{i-1}, v_{\geq})) < p(x_{i-1}, v_{\leq}) + 1 \) and \( \text{plen}(x_{i-1}) < \text{len} \) then \( \text{plen}(x_i) = \text{plen}(x_{i-1}) + 1 \). Otherwise, if \( p(x_i, v_{\geq}) < n + 1 \) then \( \text{plen}(x_i) = 1 \), else \( \text{plen}(x_i) = 0 \).

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

Proposition 2 \((\text{card}(x_i))\) If \( \text{plen}(x_i) = 1 \) then \( \text{card}(x_i) = 0 \). Otherwise, if \( p(x_i, v_{\geq}) = n + 1 \) then \( \text{card}(x_i) = \text{card}(x_{i-1}) + 1 \), else \( \text{card}(x_i) = \text{card}(x_{i-1}) \).

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

Algorithm 1 implements the lemmas with \( \text{pre}[l][0][0] = p(x_i, v_{\leq}), \text{pre}[l][0][1] = p_{\geq}(x_i, v_{\leq}), \text{pre}[l][1][1] = p(x_i, v_{\geq}), \text{pre}[l][2] = \text{plen}(x_i), \text{pre}[l][3] = \text{card}(x_i) \).

The principle of Algorithm 2 is the following. First, \( l_b = f(x) \) is computed with \( x_n \). We execute Algorithm 1 from \( x_0 \) to \( x_{n-1} \) and conversely (arrays \( \text{pre} \) and \( \text{snf} \)). We thus have for each quantity two values for each variable \( x_i \). To aggregate them, we implement regret mechanisms directly derived from Propositions 1 and 2, according to the parameters \( \text{len} \) and \( h \). Line 4 is optional but it avoids some work when the variable \( y_i \) 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 1: MINCARDS($X, \text{len}, k, h$): Integer matrix

1. $\text{pre} := \text{new Integer}[|X|][4]$ ;
2. for $l \in 0..n-1$ do
3. \hspace{1em} $\text{pre}[l][0] := \text{new Integer}[2]$ ;
4. \hspace{2em} /* Initialization from Lemma 1 */
5. \hspace{4em} if $\min(x_0) \leq k$ then
6. \hspace{5em} \hspace{1em} $\text{pre}[0][0][0] := 0$ ;
7. \hspace{5em} \hspace{1em} else
8. \hspace{6em} \hspace{1em} $\text{pre}[0][0][0] := n + 1$ ;
9. \hspace{5em} \hspace{1em} $\text{pre}[0][0][1] := n + 1$ ;
10. \hspace{4em} \hspace{1em} if $\max(x_0) > k$ then
11. \hspace{5em} \hspace{2em} $\text{pre}[0][1] := 1$ ;
12. \hspace{4em} \hspace{1em} else
13. \hspace{5em} \hspace{2em} $\text{pre}[0][1] := n + 1$ ;
14. \hspace{4em} \hspace{1em} $\text{pre}[0][2] := 0$ ;
15. \hspace{4em} \hspace{1em} $\text{pre}[0][3] := 0$ ;
16. \hspace{2em} /* Lemma 2 */
17. \hspace{4em} if $\min(x_l) \leq k$ then
18. \hspace{5em} \hspace{1em} $\text{pre}[l][0][0] := \min(\text{pre}[l-1][0][0], \text{pre}[l-1][1])$ ;
19. \hspace{5em} \hspace{1em} else
20. \hspace{6em} \hspace{1em} $\text{pre}[l][0][0] := n + 1$ ;
21. \hspace{4em} /* Lemma 3 */
22. \hspace{4em} if $\min(x_l) > k$ then
23. \hspace{5em} \hspace{1em} $\text{pre}[l][0][1] := n + 1$ ;
24. \hspace{5em} \hspace{1em} else
25. \hspace{6em} \hspace{1em} if $\text{pre}[l-1][2] \in \{0, \text{len} - 1, \text{len}\} \lor \text{pre}[l-1][3] = h$ then
26. \hspace{7em} \hspace{1em} $\text{pre}[l][0][1] := n + 1$ ;
27. \hspace{6em} \hspace{1em} else
28. \hspace{7em} \hspace{1em} $\text{pre}[l][0][1] := \min(\text{pre}[l-1][0][1], \text{pre}[l-1][0][0])$ ;
29. \hspace{4em} /* Lemma 4 */
30. \hspace{4em} if $\max(x_l) \leq k$ then
31. \hspace{5em} \hspace{1em} $\text{pre}[l][1] := n + 1$ ;
32. \hspace{5em} \hspace{1em} else
33. \hspace{6em} \hspace{1em} if $\text{pre}[l-1][2] \in \{0, \text{len}\}$ then
34. \hspace{7em} \hspace{1em} $\text{pre}[l][1] := \min(\text{pre}[l-1][1] + 1, \text{pre}[l-1][0][0] + 1)$ ;
35. \hspace{6em} \hspace{1em} else
36. \hspace{7em} \hspace{1em} $\text{pre}[l][1] := \min(\text{pre}[l-1][1], \text{pre}[l-1][0][1], \text{pre}[l-1][0][0] + 1)$
37. \hspace{4em} /* Proposition 1 */
38. \hspace{4em} if $\min(\text{pre}[l-1][0][1], \text{pre}[l-1][1]) < \text{pre}[l-1][0][0] + 1 \land \text{pre}[l-1][2] < \text{len}$ then
39. \hspace{5em} \hspace{1em} $\text{pre}[l][2] := \text{pre}[l-1][2] + 1$ ;
40. \hspace{4em} /* Proposition 2 */
41. \hspace{4em} if $\text{pre}[l][1] < n + 1$ then
42. \hspace{5em} \hspace{1em} $\text{pre}[l][2] := 1$ ;
43. \hspace{5em} \hspace{1em} else
44. \hspace{6em} \hspace{1em} $\text{pre}[l][2] := 0$ ;
45. \hspace{4em} if $\text{pre}[l][2] = 1$ then
46. \hspace{5em} \hspace{1em} $\text{pre}[l][3] := 0$ ;
47. \hspace{5em} \hspace{1em} else
48. \hspace{6em} \hspace{1em} if $\text{pre}[l][1] = n + 1$ then
49. \hspace{7em} \hspace{1em} $\text{pre}[l][3] := \text{pre}[l-1] + 1$ ;
50. \hspace{6em} \hspace{1em} else
51. \hspace{7em} \hspace{1em} $\text{pre}[l][3] := \text{pre}[l-1]$ ;
52. return $\text{pre}$ ;
Variables that take values less than or equal to \( \text{len} \). Note that \( \text{len} \) and \( x \) do not have to be equal to \( k \) or \( h \). We say \( \text{len} \) and \( x \) 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 \( s_{i+\text{len}_i-1} \), \( i \in \{0, 1, \ldots, n-1\} \), \( x_i \) and \( x_{i+\text{len}_i-1} \) are not neutral, and \( \text{len}_i \) is the maximal possible length of a sequence that starts at the \( i \)th position. Note that \( \text{len}_i \) does not have to be equal to \( len \) as \( s_{i+\text{len}_i-1} \) can cover at most \( h \) variables that take values less than or equal to \( k \). We call the set of these sequences \( S_X^N \).

### 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 [13, 14, 6].

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 \( s_{i+\text{len}_i-1} \), \( i \in \{0, 1, \ldots, n-1\} \), \( x_i \) and \( x_{i+\text{len}_i-1} \) are not neutral, and \( \text{len}_i \) is the maximal possible length of a sequence that starts at the \( i \)th position. Note that \( \text{len}_i \) does not have to be equal to \( len \) as \( s_{i+\text{len}_i-1} \) can cover at most \( h \) variables that take values less than or equal to \( k \). We call the set of these sequences \( S_X^N \).
Example 2 Consider $X = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$ and SPRINGY_FOCUS($X, [3, 3], 3, 1, 0$) 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 Figure 2(a). Hence, $S_X = \{s_{0.2}, s_{5.7}, s_{6.8}, s_{7.8}, s_{8.8}\}$. 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 SPRINGY_FOCUS constraint without the disjointness requirement SPRINGY_FOCUS_OVERLAP. More formally we define SPRINGY_FOCUS_OVERLAP 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 SPRINGY_FOCUS_OVERLAP($X, y_c, len, h, k$) iff there exists a set $S_X \subseteq S_X^y$ 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. \( \forall s_{i,j} \in S_X, j - i + 1 \leq \text{len}, x_i > k \) and \( x_j > k \)
4. \( \forall s_{i,j} \in S_X, |\{l \in s_{i,j}, x_l \leq k\}| \leq h \)

**Lemma 5** \( \text{SPRINGYFocus}(X, y_c, \text{len}, h, k) \) has a solution iff \( \text{SPRINGYFocusOverlap}(X, y_c, \text{len}, h, k) \) has a solution.

**Proof** \( \Leftarrow \) Let \( I[X \cup \{y_c\}] \) be a solution of \( \text{SPRINGYFocusOverlap} \). We order sequences in \( S_X \) by their starting points and process them in this order. Let \( s_{i,i+\text{len}_i-1} \) and \( s_{j,j+\text{len}_j-1} \) be the first two consecutive sequences in \( S_X \) that overlap. We update \( S_X \). First, we remove \( s_{j,j+\text{len}_j-1} \): \( S_X = S_X \setminus \{s_{j,j+\text{len}_j-1}\} \). Consider a sequence \( s_{i,i+\text{len}_i,j+\text{len}_j-1} \). By definition, \( x_j+\text{len}_j-1 > k \). If \( s_{i,i+\text{len}_i,j+\text{len}_j-1} \) has a prefix that contains only neutral variables then we cut it from the sequence and obtain \( s_{i',j+\text{len}_j-1} \). We add this sequence to our set: \( S_X = S_X \cup \{s_{i',j+\text{len}_j-1}\} \). So, we cut the prefix of \( s_{j,j+\text{len}_j-1} \) 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 \( \text{SPRINGYFocus} \). We extend each sequence to its maximal length to the right. This gives a solution of \( \text{SPRINGYFocusOverlap} \). \( \square \)

**Example 3** Consider \( \text{SPRINGYFocusOverlap} \) from Example 2. \( S_X = \{s_{0,2}, s_{5,7}, s_{7,8}\} \) is a possible solution (dashed lines in Figure 2(a)). 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, \( S_X = (S_X \cup \{s_{8,8}\}) \setminus \{s_{7,8}\} = \{s_{0,2}, s_{5,7}, s_{8,8}\} \).

Thanks to Lemma 5 we build an ILP reformulation for \( \text{SPRINGYFocusOverlap} \), solve this ILP and transform to a solution of the \( \text{SPRINGYFocus} \) constraint. We introduce one Boolean variable \( sv_i \) for each sequence in \( S_X \). We can write an integer linear program:

\[
\text{Minimize } \sum_{i : s_{i,i+\text{len}_i} \in S_X} sv_i \tag{1}
\]

\[
\sum_{s_{i,i+\text{len}_i} \in S_X} sv_i \geq 1 \forall x_j \in P_k \tag{2}
\]

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

**Lemma 6** \( \text{SPRINGYFocusOverlap}(X, y_c, \text{len}, h, k) \) is satisfiable if and only if 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 Equations 1–3 has a solution \( I[sv] \). We define \( S = \{s_{i,i+\text{len}_i} | sv_i = 1\} \). 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 $\text{SPRINGYFocusOverlap}(X, y_c, \text{len}, h, k)$.

$\Rightarrow$ Suppose there is a solution of the $\text{SPRINGYFocusOverlap}(X, y_c, \text{len}, h, k)$ 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 $s_{i,i+h-1} \in S_X$. This assignment satisfies Equations 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 [20]. We consider the transformation on $\text{SPRINGYFocus}$ from Example 5. This transformation is similar to the one used to propagate the $\text{SEQUENCE}$ constraint [6].

Example 4 Consider $\text{SPRINGYFocus}$ from Example 2. We build an ILP that corresponds to an equivalent $\text{SPRINGYFocusOverlap}$ constraint using Equations 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 $s_{4,6}$:

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

$$sv_0 \geq 1$$

$$sv_5 \geq 1$$

$$sv_5 + sv_6 + sv_7 \geq 1,$$

$$sv_6 + sv_7 + sv_8 \geq 1,$$

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

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

$$sv_0 - z_0 = 1$$

$$sv_5 - z_1 = 1$$

$$sv_5 + sv_6 + sv_7 - z_2 = 1,$$

$$sv_6 + sv_7 + sv_8 - z_3 = 1,$$

In matrix form, this is:

$$\begin{pmatrix}
1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & -1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & -1
\end{pmatrix} \begin{pmatrix}
sv_0 \\
sv_5 \\
z_0 \\
z_1 \\
z_2 \\
z_3
\end{pmatrix} = \begin{pmatrix}1 \\
\vdots \\
1
\end{pmatrix}$$
We append a row of zeros to the matrix and subtract the \(i\)th row from \((i + 1)\)th row for \(i = 1\) to 4. These operations do not change the set of solutions. This gives:

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & -1 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & -1 \\
0 & -1 & 0 & 1 & 0 & 0 & -1 \\
0 & 0 & -1 & -1 & 0 & 0 & 1 \\
\end{pmatrix}
\begin{pmatrix}
sv_0 \\
sv_8 \\
z_{0} \\
z_{3} \\
\vdots \\
\vdots \\
\end{pmatrix}
= 
\begin{pmatrix}
1 \\
0 \\
0 \\
0 \\
-1 \\
\end{pmatrix}
\]

The corresponding network flow graph is shown in Figure 2(b). 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 \(m\)th 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 SPRINGYFOCUS\((X, y_c, len, h, k)\). A shortest path from 0th to \(m\)th node can be found in \(O(n)\) time.

**Proof** We show that there exists a shortest path from 0th to \(m\)th 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 \(i\)th node, \(i \in \{0, \ldots, m - 1\}\) corresponds to the \(i\)th penalizing variable in the constraint and a sequence that starts at a penalizing variable is in \(S_X\).

Let \(\pi\) 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 \(\pi\): \(\pi = (0, \ldots, j, i', \ldots, g, f, \ldots, m)\), 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 \(\pi\) to \(\pi = (0, \ldots, j, i', g', \pi', f, \ldots, m)\), where \((g', \pi', f)\) is a path that uses backward arcs to reach \(f\) from \(g'\) if \((g', f) \notin G\). As the weight \(\pi'\) is 0, the weight of the updated path \(\pi\) 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 \(m\)th node. As we know that there exists a shortest path that only uses forward arcs the greedy algorithm is optimal. \(\Box\)
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

Fig. 3 The same initial configuration of Figure 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 $z_c = 2$.

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 Figure 3 shows.

**Definition 9** Let $y_c$ 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($X, y_c, len, k, z_c$) 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 in WEIGHTEDFOCUS 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 \( \text{cst}(s_{i,j}) \) is defined as \( \text{cst}(s_{i,j}) = |\{p | x_p \in U_k, x_p \in s_{i,j}\}| \). We denote cost of \( S_X \), \( \text{cst}(S_X) \), the sum \( \text{cst}(S_X) = \sum_{s_{i,j} \in S_X} \text{cst}(s_{i,j}) \). Given \( I_X \) we consider \( |P_k| = |\{x_i \in P_k\}| \). We have: \( \sum_{s_{i,j} \in S} \text{cst}(s_{i,j}) = 0 \).

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 \( X = [x_0, x_1, \ldots, x_7] \) with domains \( [1, \{0,1\}, 1, 1, \{0,1\}, 1] \) and WEIGHTEDFOCUS(\( X, [2, 3], 0, [0, 6] \)), 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(\( X, [2, 3], 0, [7, 7] \)), 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. \( S_{1,5}^1 = \{s_{0,2}, s_{3,5}\} \) or \( S_{1,5}^2 = \{s_{0,4}, s_{3,5}\} \) 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 \( s_{3,5} \) is shorter compared to the length of the last sequence in \( S_{1,5}^1 \), 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 $\infty$ 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.
Increase of cost for each set, last

In other words, as in a comparison operation between sets, we compare by the cardi-

Alternatively, from \( S_{c,j} \) by interrupting \( \text{last}(S_{c,j-1}) \). This is encoded in line 12 of

If \( x_j \) is 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, \( \text{last}(S_{c,j}) \) and \( \text{last}(S_{c',j'}) \).

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

1. \( \text{for} \ c \in \{-1, \ldots, l\} \) \( \text{do} \)

2. \( \text{for} \ j \in \{0, \ldots, n-1\} \) \( \text{do} \)

3. \( f_{c,j} \leftarrow \{\infty, \infty\}; \)

4. \( f_{0, -1} \leftarrow \{0, 0\}; \)

5. \( \text{for} \ j \in \{0, \ldots, n-1\} \) \( \text{do} \)

6. \( \text{for} \ c \in \{0, \ldots, l\} \) \( \text{do} \)

7. \( \text{if} \ x_j \in P_k \text{ then} \) /* penalizing */

8. \( \text{if} \ (l_{c,j-1} \in [1, \text{len}]) \lor (q_{c,j-1} = \infty) \text{ then} \)

9. \( f_{c,j} \leftarrow \{q_{c,j-1}, l_{c,j-1} + 1\}; \)

10. \( \text{else} \ f_{c,j} \leftarrow \{q_{c,j-1} + 1, 1\}; \)

11. \( \text{if} \ x_j \in U_k \text{ then} \) /* undetermined */

12. \( \text{if} \ (l_{c-1,j-1} \in [1, \text{len}] \land q_{c-1,j-1} = q_{c,j-1}) \lor (q_{c,j-1} = \infty) \text{ then} \)

13. \( f_{c,j} \leftarrow \{q_{c-1,j-1}, l_{c-1,j-1} + 1\}; \)

14. \( \text{else} \ f_{c,j} \leftarrow \{q_{c,j-1}, \infty\}; \)

15. \( f_{c,j} \leftarrow \{q_{c,j-1}, \infty\}; \)

16. return \( f_1; \)

![Fig. 4 Representation of one step of Algorithm 3.](image)

Algorithm 3 gives pseudocode for the propagator. The intuition behind the algo-

If \( x_j \) is 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 \( \text{last}(S_{c,j-1}) \) or start a new sequence if it is possible. This is encoded in lines 9 and 10 of the algorithm. Figure 4(a) gives a schematic representation of these arguments.

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

If \( x_j \) is in \( N_k \) then we do not increase the cost of \( S_{c,j} \) compared to \( S_{c,j-1} \). More-

First we prove a property of the dynamic programming table. We define a com-

For \( x_j \) in \( P_k \), we increase the cost of \( S_{c,j} \) as the

If \( x_j \) in \( P_k \), we do not increase the cost of \( S_{c,j} \) compared to \( S_{c,j-1} \) as the

If \( x_j \) is in \( U_k \), we have two options. We can obtain \( S_{c,j} \) from \( S_{c-1,j-1} \) by increasing

\( f_{c,j} \leftarrow \{q_{c,j-1}, \infty\}; \)

If \( x_j \) in \( U_k \), we have two options. We can obtain \( S_{c,j} \) from \( S_{c-1,j-1} \) by increasing

If \( x_j \) in \( U_k \), we have two options. We can obtain \( S_{c,j} \) from \( S_{c-1,j-1} \) by increasing

If \( x_j \) in \( U_k \), we have two options. We can obtain \( S_{c,j} \) from \( S_{c-1,j-1} \) by increasing

If \( x_j \) in \( U_k \), we have two options. We can obtain \( S_{c,j} \) from \( S_{c-1,j-1} \) by increasing

If \( x_j \) in \( U_k \), we have two options. We can obtain \( S_{c,j} \) from \( S_{c-1,j-1} \) by increasing

If \( x_j \) in \( U_k \), we have two options. We can obtain \( S_{c,j} \) from \( S_{c-1,j-1} \) by increasing
First, we prove two technical results.

**Lemma 8** Consider \( \text{WEIGHTEDFOCUS}(x_0, \ldots, x_{n-1}, y_c, \text{len}, k, z_c) \). 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 \( c, c', c'' \), \( 0 \leq c < c' < c'' \leq z_c^U \), 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 \( c, c', c'' \), \( 0 \leq c < c' < c'' \leq z_c^U \), 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, \( q_{c,j} = \min(q_{c,j-1}, q_{c,j-1}), q_{c',j} = \min(q_{c',j-1}, q_{c',j-1}) \) and \( q_{c'',j} = \min(q_{c'',j-1}, q_{c'',j-1}) \). As \( f_{c,j} \) is dummy, 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 \( f_{c',j-1} \) and \( f_{c'',j-1} \) is non-dummy. We know that \( c-1 < c' - 1 < c' \leq c'' - 1 < c'' \) 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 15, \( q_{c,j} = q_{c,j-1}, q_{c',j} = q_{c',j-1} \) 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. \( \square \)

**Proposition 4** Consider \( \text{WEIGHTEDFOCUS}(x_0, \ldots, x_{n-1}, y_c, \text{len}, k, z_c) \). 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 \((l_{-1,j-1} \in [1, \text{len}) \land q_{-1,j-1} = q_{0,j-1}) \lor (q_{0,j-1} = \infty) \) 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 13, \( f_{0,j} \) 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 15, \( f_{0,j} \) is non-dummy. \( \square \)

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

**Lemma 9** Consider \( \text{WEIGHTEDFOCUS}(X, y_c, \text{len}, k, z_c) \). 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, \( c \in [0,z_U] \). Suppose the statement holds for \( j \) 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 penatizing variable, an undetermined variable or a neutral variable.

Case 1. Consider the case \( x_j \in P_n \). Suppose \( q_{c,j-1} = \infty \) then \( q_{c+1,j-1} = \infty \) by the induction hypothesis. Hence, by Algorithm 3, lines 9 and 10, \( f_{c,j} \) 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 \( q_{c+1,j-1} < q_{c,j} < q_{c+1,j-1} + 1 \). 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 \( q_{c+1,j-1} = q_{c,j-1} < q_{c+1,j} \leq q_{c+1,j-1} + 1 \). 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 \( l_{c,j} \in [1,\text{len}] \) by Algorithm 3 line 9. As \( q_{c+1,j} = q_{c+1,j-1} + 1 \) then \( l_{c+1,j-1} \in [\text{len},\infty) \) 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 \( q_{c+1,j-1} = q_{c,j-1} < q_{c+1,j-1} + 1 \). Therefore, either \( q_{c,j} = q_{c,j-1} \) \( \land \) \( q_{c+1,j} = q_{c+1,j-1} \) or \( q_{c,j} = q_{c,j-1} + 1 \) \( \land \) \( q_{c+1,j} = q_{c+1,j-1} + 1 \). If \( q_{c,j} = q_{c,j-1} \) and \( q_{c+1,j} = q_{c+1,j-1} \) then \( l_{c,j} = l_{c,j-1} \in [1,\text{len}) \) and \( l_{c+1,j} = l_{c+1,j-1} \in [1,\text{len}) \) 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 \( l_{c,j} < l_{c+1,j} \).

Case 2. Consider the case \( x_j \in U_n \). Suppose \( q_{c,j-1} = \infty \) 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 \( q_{c+1,j-1} \leq q_{c+1,j} \) and \( q_{c,j-1} \leq q_{c+1,j-1} \). By induction hypothesis, \( q_{c+1,j-1} \leq q_{c,j-1} \leq q_{c,j-1} \). Hence, if \( q_{c,j} \) then \( q_{c,j-1} \leq q_{c,j} \leq q_{c+1,j} \). 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 \( q_{c,j-1} = q_{c+1,j-1} \) (line 12). We also know \( q_{c,j-1} \leq q_{c,j} \leq q_{c+1,j} \leq q_{c+1,j-1} \) from Case 1a. Putting everything together, we get \( q_{c,j-1} \leq q_{c,j} \leq q_{c+1,j} \). This leads to a contradiction.

– **Case 2d.** Suppose \( q_{c,j} = q_{c+1,j}, l_{c,j} < l_{c+1,j} \) and \( q_{c,j-1} = q_{c+1,j-1} \). Hence, \( q_{c,j-1} \leq q_{c,j} \leq q_{c+1,j} \). This contradicts our assumption \( q_{c,j} \).

Consider two subcases. Suppose \( q_{c,j-1} < q_{c,j-1} \). Then \( l_{c,j} = \infty \) (line 13). Hence, our assumption \( l_{c,j} < l_{c+1,j} \) is false. Suppose \( q_{c,j-1} = q_{c,j-1} \). If \( l_{c,j-1} = \infty \) then \( l_{c,j} = \infty \) (line 13).

Consider two subcases. Suppose \( q_{c,j-1} < q_{c,j-1} \). If \( l_{c,j-1} = \infty \) then \( l_{c,j} = \infty \) (line 13).

Consider two subcases. Suppose \( q_{c,j-1} \neq q_{c+1,j-1} \). Hence, \( q_{c,j-1} \leq q_{c,j} \leq q_{c+1,j} \leq q_{c+1,j-1} \). As we know from Case 1a \( q_{c+1,j-1} \leq q_{c,j} \). Hence, \( q_{c,j-1} \leq q_{c,j} \leq q_{c+1,j} \). This contradicts our assumption \( q_{c,j} \).

**Case 3.** Consider the case \( x_j \in N_k \). This case follows immediately from Algorithm 3, line 15, and the induction hypothesis.

\[ \square \]

**Lemma 10** Consider WEIGHTEDFOCUS \((X, y, len, k, z_c)\). The dynamic programming table \( f_{c,j} = \{q_{c,j}, l_{c,j} \}, c \in [0, z_c], \) \( j = 0, \ldots, n-1 \), 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 \( S_{c,j-1} \), \( c \in [0, z_c^U] \).

The first option is to cover \( x_j \). Hence, we need to extend \( last(S_{c,j-1}) \). Note that we should not start a new sequence if \( last(S_{c,j-1}) = len \) 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 \( \text{last}(S_{c,j-1}) \).

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 \( \text{last}(S_{c,j-1}) \).

Suppose \( |S_{c,j-1}| = |S_{c-1,j-1}| \) and \( |\text{last}(S_{c,j-1})| \leq |\text{last}(S_{c-1,j-1})| \).

If \( |\text{last}(S_{c-1,j-1})| < \text{len} \) then it is optimal to extend \( \text{last}(S_{c-1,j-1}) \). If \( |\text{last}(S_{c-1,j-1})| = \text{len} \) then it is optimal to interrupt \( \text{last}(S_{c,j-1}) \), 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 \( \text{last}(S_{c,j-1}) \). If \( S_{c,j-1} \) does not exist then \( S_{c,j} \) does not exist. The algorithm performs this extension in line 15. 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 \( f(c,j) \).

\( \square \)

**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 \( S_{2,7} = \{\overline{0,3}, \overline{6,7}\} \) such that \( |S_{2,7}| = 2 \).

<table>
<thead>
<tr>
<th>( c )</th>
<th>( D(x_0) )</th>
<th>( D(x_1) )</th>
<th>( D(x_2) )</th>
<th>( D(x_3) )</th>
<th>( D(x_4) )</th>
<th>( D(x_5) )</th>
<th>( D(x_6) )</th>
<th>( D(x_7) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( [1,1] )</td>
<td>( [0,1] )</td>
<td>( [1,1] )</td>
<td>( [0,1] )</td>
<td>( [1,1] )</td>
<td>( [0,1] )</td>
<td>( [1,1] )</td>
<td>( [0,1] )</td>
</tr>
<tr>
<td>1 ( z_c^U = 2 )</td>
<td>( [1,2] )</td>
<td>( [2,1] )</td>
<td>( [2,1] )</td>
<td>( [2,2] )</td>
<td>( [1,4] )</td>
<td>( [2,1] )</td>
<td>( [2,2] )</td>
<td>( [2,3] )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( c )</th>
<th>( D(x_0) )</th>
<th>( D(x_1) )</th>
<th>( D(x_2) )</th>
<th>( D(x_3) )</th>
<th>( D(x_4) )</th>
<th>( D(x_5) )</th>
<th>( D(x_6) )</th>
<th>( D(x_7) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( [1,1] )</td>
<td>( [0,1] )</td>
<td>( [1,1] )</td>
<td>( [0,1] )</td>
<td>( [1,1] )</td>
<td>( [0,1] )</td>
<td>( [1,1] )</td>
<td>( [0,1] )</td>
</tr>
<tr>
<td>1 ( z_c^U = 2 )</td>
<td>( [1,2] )</td>
<td>( [2,1] )</td>
<td>( [2,1] )</td>
<td>( [2,2] )</td>
<td>( [1,4] )</td>
<td>( [2,1] )</td>
<td>( [2,2] )</td>
<td>( [2,3] )</td>
</tr>
</tbody>
</table>

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

### 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(\( X, y_c, \text{len}, k, z_c \)). **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 \( O(z_{v}^{l}) \) pairs of values \( f_{c_1,i-1} \) and \( b_{c_2,n-i-2} \), \( c_1,c_2 \in [0,z_{v}^{l}] \) which are neighbour columns to the \( i \)th column. It is easy to show that if we consider all possible pairs of elements in \( f_{c_1,i-1} \) and \( b_{c_2,n-i-2} \) then we determine if there exists a support for \( x_i = v \). There are \( O(z_{v}^{l} \times z_{v}^{l}) \) such pairs. The main part of the proof shows that it sufficient to consider \( O(z_{v}^{l}) \) 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 \( O(z_{v}^{l}) \) pairs of values \( f_{c_1,i-1} \) and \( b_{c_2,n-i-2} \), \( c_1,c_2 \in [0,z_{v}^{l}] \). 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 \( i \)th 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 \( b_{c_2,n-i-2} \), \( c_1,c_2 \in [0,z_{v}^{l}] \) 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 \([x_{0},...,x_{j-1}]\) and the suffix \([x_{j+1},...,x_{n-1}]\), 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. \( \text{cst}(S) = c_1 + c_2 + 1 \) and \( |S| = |S_{c_1,i-1}| + |S_{c_2,n-i-2}| + 1 \).
- The second option is to extend \( \text{last}(S_{c_1,i-1}) \) to the right by one if \( |\text{last}(S_{c_1,i-1})| < \text{len} \). 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: \( \text{cst}(S) = c_1 + c_2 + 1 \) and \( |S| = |S_{c_1,i-1}| + |S_{c_2,n-i-2}| \).
- The third option is to extend \( \text{last}(S_{c_2,n-i-2}) \) to the left by one if \( |\text{last}(S_{c_2,n-i-2})| < \text{len} \). This case is symmetric to the previous case.

The fourth and the cheapest option is to glue \( \text{last}(S_{c_1,i-1}), x_v \) and \( \text{last}(S_{c_2,n-i-2}) \) to a single sequence if \( |\text{last}(S_{c_1,i-1})| + |\text{last}(S_{c_2,n-i-2})| < \text{len} \). Hence, \( S'_{c_1,i-1} = S_{c_1,i-1} \setminus \text{last}(S_{c_1,i-1}), S'_{c_2,n-i-2} = S_{c_2,n-i-2} \setminus \text{last}(S_{c_2,n-i-2}) \) and \( s' \) is a concatenation of \( \text{last}(S_{c_1,i-1}), x_v \) and \( \text{last}(S_{c_2,n-i-2}) \). Then the union \( S = S'_{c_1,i-1} \cup S'_{c_2,n-i-2} \cup \{s'\} \) forms a cover: \( \text{cst}(S) = c_1 + c_2 + 1 \) 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 \( b_{c_2,n-i-2} \), \( c_1,c_2 \in [0,z_{v}^{l}] \) and check the four cases above. If obtained cover \( S \) is such that \( \text{cst}(S) \leq z_{v}^{l} \) and \( |S| \leq \max(y_v) \) 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 \( b_{c_2,n-i-2} \), \( c_1,c_2 \in [0,z_{v}^{l}] \) then finding a support takes \( O((z_{v}^{l})^2) \) 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 \( \text{last}(S_{c_2,n-i-2}) < \text{last}(S_{c_2,n-i-2}) \). 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 \( O(z_c^U) = O(\max(z_e)) \) 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: \( \text{cst}(S) = c_1 + c_2 \) 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, \( |S_{c_1,i-1}| + |S_{c_2,n-i-2}| \leq |S_{c_1,i-1}| + |S_{c_2,n-i-2}| \leq \max(y_c) + 1 \). In the case of equality, we know that \( \text{last}(S_{c_2,n-i-2}) < \text{last}(S_{c_2,n-i-2}) \). 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 \( v_2, v_1 < k \) and \( v_2 > k \) in \( D(x_i) \) in \( O(\max(z_e)) \) time for each variable \( x_i, i = 0, \ldots, n - 1 \). Hence, the time complexity to enforce domain consistency is \( O(n \max(z_e)) \).

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 \( c_1 + c_2 \leq z_U^c \). □

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 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, \( |\text{last}(S_{0,3})| + |\text{last}(S_{1,5})| = 2 + 3 = \text{len} = 5 \). Therefore, we cannot merge last\( (S_{0,3}), x_4 = 1 \) 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 \( |\text{last}(S_{1,3})| + |\text{last}(S_{0,5})| = 4 + 1 = \text{len} = 5 \), we cannot merge last\( (S_{1,3}), x_4 = 1 \) 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 \( y_{c,n-1} < \max(y_c) \) 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 \( c \text{st}(S_X) \) and \( |S_X| \) by at most one.

Decomposition with \( O(n) \) variables and constraints. Alternatively we can decompose \textsc{WeightedFocus} using \( O(n) \) additional variables and constraints.

Given \textsc{Focus}\((X, y_c, \text{len}, k)\), 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 \textsc{WeightedFocus} as follows:

\textsc{WeightedFocus}(\(X, y_c, \text{len}, k, z_c\)) ⇐ \textsc{Focus}(\(X, y_c, \text{len}, k\)) \∧ (∀l, 0 ≤ l < n, [(x_l ≤ k) \∧ (b_l = 0)] \∨ [(x_l > k) \∧ (b_l = 1)]) \∧ \sum_{l \in \{0, 1, \ldots, n-1\}} b_l ≤ z_c.

Enforcing BC on each constraint of the decomposition is weaker than BC on \textsc{WeightedFocus}. Given \( x_i \in X \), a value may have a unique support for \textsc{Focus} which violates \( \sum_{l \in \{0, 1, \ldots, n-1\}} b_l ≤ z_c \), and conversely. Consider \( n=5 \)}
Consider variables $X = [x_0, x_1, \ldots, x_5]$ with domains $[1, \{0, 1\}, 1, \{0, 1\}, 1]$ and $\text{WEIGHTEDFOCUS}(X, [2, 3], 0, [0, 4])$. Following approach in Section 3.3, we consider six sequences $S_X = \{s_{00}, s_{13}, s_{24}, s_{35}, s_{46}, s_{56}, s_{66}\}$. The cost of any solution that uses sequences from $S_X$ is 6. However, there exists a solution of $\text{WEIGHTEDFOCUS}$ with cost 4: $S_X = \{s_{01}, s_{23}, s_{55}\}$, $y_c = 3$ and $z_c = 4$.

5 Weighted Springy FOCUS

We consider a further generalization of the FOCUS constraint that combines SPRINGYFOCUS 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 $\text{WEIGHTEDSPRINGYFOCUS}(X, y_c, len, h, k, z_c)$ 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. $\sum_{s_{i,j} \in S} |s_{i,j}| = \sum_{s_{i,j} \in S} \text{cost}(s_{i,j}) + |P_k|$. However, $\text{cost}(s_{i,j})$ is the number of undetermined and neutral variables covered $s_{i,j}$, $\text{cost}(s_{i,j}) = |\{p | x_p \in U_k \cup N_k, x_p \in s_{i,j}\}|$ 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 \(S_{c,j}^1\) and \(S_{c,j}^2\) of minimum cardinality. The issue here is how to compare \(\text{last}(S_{c,j}^1)\) and \(\text{last}(S_{c,j}^2)\) if they cover a different number of neutral variables. Luckily, we can avoid this problem due to the following monotonicity property. If \(\text{last}(S_{c,j}^1)\) and \(\text{last}(S_{c,j}^2)\) are not equal to infinity then they both end at the same position \(j\). Hence, if \(\text{last}(S_{c,j}^1) \preceq \text{last}(S_{c,j}^2)\) then the number of neutral variables covered by \(\text{last}(S_{c,j}^1)\) is no larger than the number of neutral variables covered by \(\text{last}(S_{c,j}^2)\). Therefore, we can define order on sets \(S_{c,j}\) as we did in Section 4 for \textsc{WeightedFocus}.

Our bounds disentailment detection algorithm for \textsc{WeightedSpringYFocus} mimics Algorithm 3. We show a pseudocode for it in Algorithm 4.

**Algorithm 4: \textsc{WeightedSpringYFocus}(x_0, \ldots, x_{n-1})**

```plaintext
for c ∈ [−z_{c,0}, \ldots, z_{c,n-1}]
do
  for j ∈ [−1, n − 1]
do
    \(f_{c,j} \leftarrow \{\infty, \infty, \infty\};\)
  \(f_{0,−1} \leftarrow \{0, 0, 0\};\)
  for j ∈ [0, n − 1]
do
    for c ∈ [0, j]
do
      if \(x_j \in P_k\) /* penalizing */
        \(f_{c,j} \leftarrow \{q_{c,j−1} = \infty\};\)
      then
        \(f_{c,j} \leftarrow \{q_{c,j−1}, l_{c,j−1} + 1, h_{c,j−1}\};\)
      if \(x_j \in U_k\) /* undetermined */
        \(f_{c,j} \leftarrow \{q_{c,j−1} = 1, 1, 0\};\)
      then
        \(f_{c,j} \leftarrow \{q_{c,j−1}, l_{c,j−1} + 1, h_{c,j−1}\};\)
      if \(x_j \in N_k\) /* neutral */
        \(f_{c,j} \leftarrow \{q_{c,j−1}, \infty, \infty\};\)
      then
        \(f_{c,j} \leftarrow \{q_{c,j−1}, l_{c,j−1} + 1, h_{c,j−1} + 1\};\)
      else
        \(f_{c,j} \leftarrow \{q_{c,j−1}, l_{c,j−1} + 1, h_{c,j−1}\};\)
      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 \(f_{c,j}, c \in [0, z_{c,0}], j \in \{0, 1, \ldots, n − 1\}\), where \(z_{c,0} = \max(z_c) - |P_k|\), 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 \(\text{last}(S_{c,j})\). 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 $\text{cst}(S_{c-1,j-1})$ by one and increasing the number of covered neutral variables by $\text{last}(S_{c,j-1})$ (Figure 4(c), the gray arc). Alternatively, we can obtain $S_{c,j}$ from $S_{c,j-1}$ by interrupting $\text{last}(S_{c,j-1})$ (Figure 4(c), the black arc). BC can be enforced using two modifications of the corresponding algorithm for WEIGHTEDFOCUS

Lemma 12 Consider WEIGHTEDSPRINGYFOCUS$(X, y_c, \text{len}, h, k, z_c)$. 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 $\text{last}(S_{c,i-1})$, $x_v$ and $\text{last}(S_{c,i-2})$ to a single sequence if $|\text{last}(S_{c,i-1})| + |\text{last}(S_{c,i-2})| < \text{len}$ and $h(\text{last}(S_{c,i-1})) + h(\text{last}(S_{c,i-2})) \leq h$. Hence, $S'_{c,i-1} = S_{c,i-1} \setminus \text{last}(S_{c,i-1})$, $S'_{c,i-2} = S_{c,i-2} \setminus \text{last}(S_{c,i-2})$ and $S'$ is a concatenation of $\text{last}(S_{c,i-1}), x_v$ and $\text{last}(S_{c,i-2})$. Then the union $S = S'_{c,i-1} \cup S'_{c,i-2} \cup \{s'\}$ forms a cover: $\text{cst}(S) = c_1 + c_2 + 1$, $|S| = |S_{c,i-1}| + |S_{c,i-2}| - 1$ and $h(S) = h(\text{last}(S_{c,i-1})) + h(\text{last}(S_{c,i-2})).$

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 $S_{c,i-1}, x_v$ and $S_{c,i-2}$. Hence, if $c_1 + c_2 + 1 \leq z_{c,v}$, $|\text{last}(S_{c,i-1})| + |\text{last}(S_{c,i-2})| < \text{len}$ and $h(\text{last}(S_{c,i-1})) + h(\text{last}(S_{c,i-2})) < h$ 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 $\tau_0$ and $\tau_1$ (in addition to the initial state) as follows. The state $\tau_0$ captures all values $v \leq k$ not included in any subsequence in $S_X$. The set of states $\tau_1$ captures the values belonging to a subsequence in $S_X$. The transition between $\tau_0$ and $\tau_1$ is quite straightforward following the semantic of WEIGHTEDSPRINGYFOCUS, however, the automaton is non-deterministic as on seeing $v \leq k$ in $\tau_1$, it either covers the variable or interrupts the last sequence. The automaton needs 3 counters to compute $\text{len}$, $y_c$ 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 automata. 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 $i$th sub-sequence. $D(S_i) = \{0, \ldots, n + \max(y_c)\}$; $E_i$ the end of the $i$th sub-sequence. $D(E_i) = \{0, \ldots, n + \max(y_c)\}$; $T_j$ the index of the subsequence in $S_X$ containing $x_j$. $D(T_j) = \{0, \ldots, \max(y_c)\}$; $Z_i$ the index of the subsequence in $S_X$ containing $x_j$ s.t. the value of $x_j$ is less than or equal to $k$. $D(Z_i) = \{0, \ldots, \max(y_c)\}$; $last_c$ the cardinality of $S_X$. $D(last_c) = \{0, \ldots, \max(y_c)\}$; $Card$, a vector of $\max(y_c)$ variables having $\{0, \ldots, h\}$ as domains.

**WeightedSpringyFocus**($X, y_c, len, h, k, z_c$) ⇔

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

$\begin{array}{ll}
last_c \leq y_c; & GCC([T_0, \ldots, T_{n-1}], \{0\}, [n-z_c]); \\
& GCC([Z_0, \ldots, Z_{n-1}], \{1, \ldots, \max(y_c)\}, Card);
\end{array}$

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.
Three Generalizations of the FOCUS Constraint

The 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 behavior 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 Région’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 Figure 5. We use the procedure \(getColumn(\text{Integer}[][], m, k)\) for extracting the \(k\)th column of the matrix given as argument.

**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. 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.

Fig. 5 Model of the SLS benchmark.

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 ≤ 7 (instances n-1), h ≤ 6 (n-2) and h ≤ 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.\footnote{Recall that the average CPU includes only the runtime of the successful runs.}
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.

| Table 3 SLS with \texttt{WEIGHTEDFOCUS} and its decomposition using \texttt{DomOverWDeg}. |

<table>
<thead>
<tr>
<th>Tag</th>
<th>(16_1)</th>
<th>(16_2)</th>
<th>(16_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(#sol)</td>
<td>\texttt{avg.}</td>
<td>\texttt{dev.}</td>
<td>\texttt{Nodes}</td>
</tr>
<tr>
<td>F</td>
<td>49</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>D1</td>
<td>49</td>
<td>3.6</td>
<td>9.5</td>
</tr>
<tr>
<td>F</td>
<td>49</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>D1</td>
<td>49</td>
<td>3.6</td>
<td>9.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>(20_1)</th>
<th>(20_2)</th>
<th>(20_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(#sol)</td>
<td>\texttt{avg.}</td>
<td>\texttt{dev.}</td>
<td>\texttt{Nodes}</td>
</tr>
<tr>
<td>F</td>
<td>49</td>
<td>7.5</td>
<td>81</td>
</tr>
<tr>
<td>D1</td>
<td>43</td>
<td>14.1</td>
<td>35</td>
</tr>
</tbody>
</table>

| Table 4 SLS with \texttt{WEIGHTEDFOCUS} and its decomposition using a static branching. |

<table>
<thead>
<tr>
<th>Tag</th>
<th>(16_1)</th>
<th>(16_2)</th>
<th>(16_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(#sol)</td>
<td>\texttt{avg.}</td>
<td>\texttt{dev.}</td>
<td>\texttt{Nodes}</td>
</tr>
<tr>
<td>F</td>
<td>23</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D1</td>
<td>21</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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.

**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 \textsc{WeightedSpringFOCUS}(X, \( y_c \), len, \( h \), 0, \( z_c \)) with a set of \textsc{Among} constraints. The decision version of the problem is presented in Figure 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 Figure 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.

```plaintext
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 \textsc{WeightedSpringFOCUS}
MODEL:
IntVar[] X;  // size: n, domain \{0, 1\}
IntVar y_c, z_c;  // used for \textsc{WeightedSpringFOCUS}
\forall d \in 1..m, l[d] \leq \sum_{i=s[d]}^{e[d]} X[i] \leq u[d];  // the set of \textsc{Among} constraints
\textsc{WeightedSpringFOCUS}(X, y_c, len, h, 0, z_c);
```

*Fig. 6* Model of the Cumulative Scheduling with Rentals problem

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

Figure 7 confirms the gain of flexibility illustrated by Figure 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 \textsc{Among} constraint for each \((i, j)\) s.t. \( j \geq i + 5 \) in the 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 \textit{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 between 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.

\textit{Model.} The full description and a CP model is in [12]. Figure 6.4 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 \textsc{WeightedSpringyFOCUS} and its decompositions. We generated 4 sets of instances distinguished by the numbers of
### Table 5 Scheduling with rentals using DomOverWDeg

<table>
<thead>
<tr>
<th>h=0</th>
<th>#sol</th>
<th>Time</th>
<th>Nodes</th>
<th>Nodes/s</th>
<th>#sol</th>
<th>Time</th>
<th>Nodes</th>
<th>Nodes/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>20</td>
<td>40</td>
<td>59.09</td>
<td>122</td>
<td>20</td>
<td>40</td>
<td>59.09</td>
<td>122</td>
</tr>
<tr>
<td>D</td>
<td>20</td>
<td>210</td>
<td>212</td>
<td>210</td>
<td>20</td>
<td>210</td>
<td>212</td>
<td>210</td>
</tr>
<tr>
<td>h=1</td>
<td>#sol</td>
<td>Time</td>
<td>Nodes</td>
<td>Nodes/s</td>
<td>#sol</td>
<td>Time</td>
<td>Nodes</td>
<td>Nodes/s</td>
</tr>
<tr>
<td>F</td>
<td>20</td>
<td>95</td>
<td>176</td>
<td>341394</td>
<td>20</td>
<td>95</td>
<td>176</td>
<td>341394</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>20</td>
<td>59.09</td>
<td>210</td>
<td>20</td>
<td>59.09</td>
<td>210</td>
<td>210</td>
</tr>
<tr>
<td>h=2</td>
<td>#sol</td>
<td>Time</td>
<td>Nodes</td>
<td>Nodes/s</td>
<td>#sol</td>
<td>Time</td>
<td>Nodes</td>
<td>Nodes/s</td>
</tr>
<tr>
<td>F</td>
<td>20</td>
<td>96</td>
<td>179</td>
<td>341134</td>
<td>20</td>
<td>96</td>
<td>179</td>
<td>341134</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>20</td>
<td>59.09</td>
<td>210</td>
<td>20</td>
<td>59.09</td>
<td>210</td>
<td>210</td>
</tr>
</tbody>
</table>

**INPUT:**

Int n; // number chords, indexed from 0 to n − 1
Int[ ][ ] costMatrix; // size: n × n, matrix of costs between pairs of chords
Int len, h, k; // WeightEDSpringyFOCUS

**MODEL:**

IntVar[ ] Chords; // size: n, domain {0, 1, . . . , n − 1}
IntVar[ ] Costs; // size: n − 1, domain: all possible costs
Int nChange; // threshold from which a cost is considered as high
IntVar yc, zc; // WeightEDSpringyFOCUS

∀ i ∈ 0...n − 2, TABLE(Chords[i], Chords[i + 1], Costs[i]); // cost of each pair
ALLDIFFERENT(Chords);

WeightEDSpringyFOCUS(Costs, yc, len, h, zc);

---

**Fig. 8** Model of the Sorting Chords benchmark.
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.

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