

# Three Generalizations of the FOCUS Constraint

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

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

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

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

## Abstract

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

## 1 Introduction

Many discrete optimization problems have constraints on the objective function. Being able to represent such constraints is fundamental to deal with many real world industrial problems. Constraint programming is a promising approach to express and filter such constraints. In particular, several constraints have been proposed for obtaining well-balanced solutions [?; ?; ?]. Recently, the FOCUS constraint [?] 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,  $X = [x_0, x_1, \dots, x_{n-1}]$  is a sequence of variables and  $s_{i,j}$  is a sequence of indices of consecutive variables in  $X$ , such that  $s_{i,j} = [i, i+1, \dots, j]$ ,  $0 \leq i \leq j < n$ . We let  $|E|$  be the size of a collection  $E$ .

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

FOCUS can be used in various contexts including cumulative scheduling problems where some excesses of capacity can be tolerated to obtain a solution [?]. 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 [?]. In practice, excesses might be tolerated by, for example, renting a new machine to produce more resource. Suppose the rental price decreases proportionally to its duration: it is cheaper to rent a machine during a single interval than to make several rentals. On the other hand, rental intervals have generally a maximum

possible duration. FOCUS can be set to concentrate (non null) excesses in a small number of intervals, each of length at most  $len$ .

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

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

## 2 Springy FOCUS

In Definition 1, each sequence in  $S_X$  contains *exclusively* values  $v > k$ . In many practical cases, this property is too strong. Consider one simple instance of the problem in the introduc-

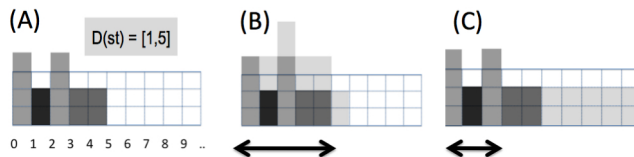


Figure 1: (A) Problem with 4 fixed activities and one activity of length 5 that can start from time 1 to 5. (B) Solution satisfying  $\text{FOCUS}(X, [1, 1], 5, 0)$ , with a new machine rented for 5 days. (C) Practical solution violating  $\text{FOCUS}(X, [1, 1], 5, 0)$ , with a new machine rented for 3 days but not used on the second day.

tion, in Figure 1, where one variable  $x_i \in X$  is defined per

point in time  $i$  (e.g., one day), to represent excesses of capacity. Initially, 4 activities are fixed and one activity  $a$  remains to be scheduled (drawing A), of duration 5 and that can start from day 1 to day 5. If  $\text{FOCUS}(X, y_c = 1, 5, 0)$  is imposed then  $a$  must start at day 1 (solution B). We have one 5 day rental interval. Assume now that the new machine may not be used every day. Solution (C) gives one rental of 3 days instead of 5. Furthermore, if  $\text{len} = 4$  the problem will have no solution using  $\text{FOCUS}$ , 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  $\text{FOCUS}$ , 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  $k, \text{len}, h$  be three integers,  $1 \leq \text{len} \leq |X|, 0 \leq h < \text{len} - 1$ . An instantiation of  $X \cup \{y_c\}$  satisfies  $\text{SPRINGYFOCUS}(X, y_c, \text{len}, h, k)$  iff there exists a set  $S_X$  of disjoint sequences of indices  $s_{i,j}$  such that four conditions are all satisfied: (1)  $|S_X| \leq y_c$  (2)  $\forall x_l \in X, x_l > k \Rightarrow \exists s_{i,j} \in S_X$  such that  $l \in s_{i,j}$  (3)  $\forall s_{i,j} \in S_X, j - i + 1 \leq \text{len}, x_i > k$  and  $x_j > k$ . (4)  $\forall s_{i,j} \in S_X, |\{l \in s_{i,j}, x_l \leq k\}| \leq h$

Bounds consistency (BC) on  $\text{SPRINGYFOCUS}$  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  $\text{FOCUS}$  [?]). Thus, we propose a BC algorithm. The first step is to traverse  $X$  from  $x_0$  to  $x_{n-1}$ , to compute the minimum possible number of disjoint sequences in  $S_X$  (a lower bound for  $y_c$ ), the *focus cardinality*, denoted  $fc(X)$ . We use the same notation for subsequences of  $X$ .  $fc(X)$  depends on  $k, \text{len}$  and  $h$ .

**Definition 3.** Given  $x_l \in X$ , we consider three quantities. (1)  $p(x_l, v_{\leq})$  is the focus cardinality of  $[x_0, x_1, \dots, x_l]$ , assuming  $x_l \leq k$ , and  $\forall s_{i,j} \in S_{[x_0, x_1, \dots, x_l]}, j \neq l$ . (2)  $\underline{p}_S(x_l, v_{\leq})$  is the focus cardinality of  $[x_0, x_1, \dots, x_l]$ , assuming  $x_l \leq k$  and  $\exists i, s_{i,l} \in S_{[x_0, x_1, \dots, x_l]}$ . (3)  $p(x_l, v_{>})$  is the focus cardinality of  $[x_0, x_1, \dots, x_l]$  assuming  $x_l > k$ .

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

**Property 1.**  $\underline{p}_S(x_0, v_{\leq}) = \underline{p}_S(x_{n-1}, v_{\leq}) = n + 1$ , and  $fc(X) = \min(\underline{p}(x_{n-1}, v_{\leq}), \underline{p}(x_{n-1}, v_{>}))$ .

*Proof.* By construction from Definitions 2 and 3.  $\square$

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

---

**Algorithm 1:**  $\text{MINCARDS}(X, \text{len}, k, h)$ : Integer matrix

---

```

1 pre := new Integer[|X|][4];
2 for l ∈ 0..n - 1 do
3   pre[l][0] := new Integer[2];
4   for j ∈ 1..3 do pre[l][j] := new Integer[1];
5   ;
6 Initialization Lemma 1.;
7 for l ∈ 1..n - 1 do Lemmas 2, 3, 4 and Propositions 1 and 2.;
8 ;
9 return pre;

```

---

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

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

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

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

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

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

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

The principle of Algorithm 2 is the following. First,  $lb = fc(X)$  is computed with  $x_{n-1}$ . We execute Algorithm 1 from  $x_0$  to  $x_{n-1}$  and conversely (arrays  $\text{pre}$  and  $\text{suf}$ ). We thus have for each quantity two values for each variable  $x_l$ . To aggregate them, we implement regret mechanisms directly derived from Propositions 2 and 1, according to the parameters  $\text{len}$  and  $h$ .

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

### 3 Weighted FOCUS

We present  $\text{WEIGHTEDFOCUS}$ , that extends  $\text{FOCUS}$  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  $S_X$ . It distinguishes between solutions that are equivalent with respect to the number of sequences in  $S_X$  but not with respect to their length, as Figure 2 shows.

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

```

1  pre := MINCARDS( $X, len, k, h$ );
2  Integer lb := min(pre[n-1][0][0], pre[n-1][1]);
3  if min( $y_c$ ) < lb then  $D(y_c) := D(y_c) \setminus \{\min(y_c), lb\}$ ;
4  ;
5  if min( $y_c$ ) = max( $y_c$ ) then
6    suf := MINCARDS( $[x_{n-1}, x_{n-2}, \dots, x_0], len, k, h$ );
7    for  $l \in 0..n-1$  do
8      if pre[l][0][0] + suf[n-1-l][0][0] > max( $y_c$ ) then
9        Integer regret := 0; Integer add := 0;
10       if pre[l][1] ≤ pre[l][0][1] then add := add + 1;
11       ;
12       if suf[n-1-l][1] ≤ suf[n-1-l][0][1] then
13         add := add + 1;
14       ;
15       if pre[l][2] + suf[n-1-l][2] - 1 ≤ len ∧
16         pre[l][3] + suf[n-1-l][3] + add - 1 ≤ h then regret
17         := 1;
18       ;
19       if
20         pre[l][0][1] + suf[n-1-l][0][1] - regret > max( $y_c$ )
21         then  $D(x_l) := D(x_l) \setminus \{\min(x_l), k\}$ ;
22       ;
23       Integer regret := 0;
24       if pre[l][2] + suf[n-1-l][2] - 1 ≤ len ∧
25         pre[l][3] + suf[n-1-l][3] - 1 ≤ h then regret := 1;
26       ;
27       if pre[l][1] + suf[n-1-l][1] - regret > max( $y_c$ ) then
28          $D(x_l) := D(x_l) \setminus \{k, \max(x_l)\}$ ;
29       ;
30     return  $X \cup \{y_c\}$ ;

```

**Definition 4.** Let  $y_c$  and  $z_c$  be two integer variables and  $k, len$  be two integers, such that  $1 \leq len \leq |X|$ . 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)  $\sum_{s_{i,j} \in S_X} |s_{i,j}| \leq z_c$ .

**Definition 5** ([?]). 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 labeled  $P_k$ , and similarly for  $U_k$  and  $N_k$ .

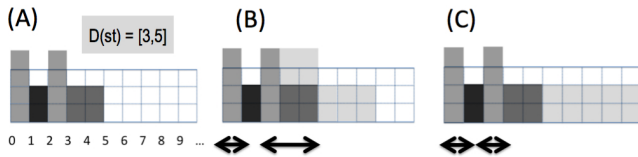


Figure 2: (A) Problem with 4 fixed activities and one activity of length 5 that can start from time 3 to 5. We assume  $D(y_c) = \{2\}$ ,  $len = 3$  and  $k = 0$ . (B) Solution satisfying WEIGHTEDFOCUS with  $z_c = 4$ . (C) Solution satisfying WEIGHTEDFOCUS with  $z_c = 2$ .

**Dynamic Programming (DP) Principle** Given a partial instantiation  $I_X$  of  $X$  and a set of sequences  $S_X$  that covers all penalizing variables in  $I_X$ , we consider two terms: the number of variables in  $P_k$  and the number of *undetermined* variables, in  $U_k$ , covered by  $S_X$ . We want to find a set  $S_X$  that minimizes the second term. Given a sequence of variables  $s_{i,j}$ , the cost  $cst(s_{i,j})$  is defined as  $cst(s_{i,j}) = \{p | x_p \in U_k, x_p \in s_{i,j}\}$ . We denote cost of  $S_X$ ,  $cst(S_X)$ ,

the sum  $cst(S_X) = \sum_{s_{i,j} \in S_X} cst(s_{i,j})$ . Given  $I_X$  we consider  $|P_k| = |\{x_i \in P_k\}|$ . We have:  $\sum_{s_{i,j} \in S} |s_{i,j}| = \sum_{s_{i,j} \in S} cst(s_{i,j}) + |P_k|$ .

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

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

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

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

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

**Ordering of sequences in  $S_{c,j}$ .** We introduce an order over sequences in  $S_{c,j}$ . Given a set of sequences in  $S_{c,j}$  we sort them by their starting points. We denote  $last(S_{c,j})$  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  $S_{c,j}$ ,  $c \in [0, \max(z_c)]$ ,  $j \in \{0, 1, \dots, n-1\}$ .** We define a comparison operation between two sets  $S_{c,j}$  and  $S_{c',j'}$ .  $S_{c,j} \leq S_{c',j'}$  iff  $|S_{c,j}| < |S_{c',j'}|$  or  $|S_{c,j}| = |S_{c',j'}|$  and  $last(S_{c,j}) \leq last(S_{c',j'})$ . 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  $[x_0, \dots, x_j]$ . 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  $[x_0, x_1, \dots, x_j]$ ,
2.  $cst(S_{c,j}) = c$ ,

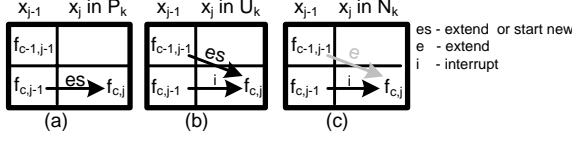


Figure 3: Representation of one step of Algorithm 3.

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  $x_0, \dots, x_j$ ,  $S_{c,j}$  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 3.** Consider WEIGHTEDFOCUS from Example 2. Consider the subsequence  $[x_0, x_1]$ .  $S_{0,1} = \{s_{0,0}\}$ ,  $S_{1,1} = \{s_{0,1}\}$ . Note that  $S_{2,1}$  does not exist. Consider the subsequence  $[x_0, \dots, x_5]$ . 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 disentanglement** Each cell in the dynamic programming table  $f_{c,j}$ ,  $c \in [0, z_c^U]$ ,  $j \in \{0, 1, \dots, n-1\}$ , 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  $f_{-1,j}$ ,  $j = -1, \dots, n-1$  to keep uniform notations.

---

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

---

```

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

---

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

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

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

If  $x_j \in P_k$  then we do not increase the cost of  $S_{c,j}$  compared to  $S_{c,j-1}$  as the cost only depends on  $x_j \in U_k$ . Hence, the best move for us is to extend  $last(S_{c,j-1})$  or start a new sequence if it is possible. This is encoded in lines 9 and 10 of the algorithm. Figure 3(a) gives a schematic representation of these arguments.

If  $x_j \in U_k$  then we have two options. We can obtain  $S_{c,j}$  from  $S_{c-1,j-1}$  by increasing  $cost(S_{c-1,j-1})$  by one. This means that  $x_i$  will be covered by  $last(S_{c,j})$ . Alternatively, from  $S_{c,j-1}$  by interrupting  $last(S_{c,j-1})$ . This is encoded in line 13 of the algorithm (Figure 3(b)).

If  $x_j \in N_k$  then we do not increase the cost of  $S_{c,j}$  compared to  $S_{c,j-1}$ . Moreover, we must interrupt  $last(S_{c,j-1})$ , line 16 (Figure 3(c), ignore the gray arc).

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

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

**Lemma 6.** Consider WEIGHTEDFOCUS( $X, y_c, len, k, z_c$ ). The dynamic programming table  $f_{c,j} = \{q_{c,j}, l_{c,j}\}$   $c \in [0, z_c^U]$ ,  $j = 0, \dots, n-1$ , 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  $O(n \max(z_c))$ .

**Example 4.** Table 1 shows an execution of Algorithm 3 on WEIGHTEDFOCUS from Example 2. Note that  $|P_0| = 5$ . Hence,  $z_c^U = max(z_c) - |P_0| = 2$ . As can be seen from the table, the constraint has a solution as there exists a set  $S_{2,7} = \{s_{0,3}, s_{5,7}\}$  such that  $|S_{2,7}| = 2$ .

**Bounds consistency** To enforce BC on variables  $x$ , 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  $x$ .

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

*Proof.* (Sketch) We build dynamic programming tables  $f$  and  $b$ . We will show that to check if  $x_i = v$  has a support it is sufficient to examine  $O(z_c^U)$  pairs of values  $f_{c_1, i-1}$  and  $b_{c_2, n-i-2}$ ,  $c_1, c_2 \in [0, z_c^U]$  which are neighbor columns to the

$i$ th column. It is easy to show that if we consider all possible pairs of elements in  $f_{c_1, i-1}$  and  $b_{c_2, n-i-2}$  then we determine if there exists a support for  $x_i = v$ . There are  $O(z_c^U \times z_c^U)$  such pairs. The main part of the proof shows that it sufficient to consider  $O(z_c^U)$  such pairs. In particular, to check a support for a variable-value pair  $x_i = v$ ,  $v > k$ , for each  $f_{c_1, i-1}$  it is sufficient to consider only one element  $b_{c_2, n-i-2}$  such that  $b_{c_2, n-i-2}$  is non-dummy and  $c_2$  is the maximum value that satisfies inequality  $c_1 + c_2 + 1 \leq z_c^U$ . To check a support for a variable-value pair  $x_i = v$ ,  $v \leq k$ , for each  $f_{c_1, i-1}$  it is sufficient to consider only one element  $b_{c_2, n-i-2}$  such that  $b_{c_2, n-i-2}$  is non-dummy and  $c_2$  is the maximum value that satisfies inequality  $c_1 + c_2 \leq z_c^U$ .  $\square$

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

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

**Proposition 4.** *Given FOCUS( $X, y_c, \text{len}, k$ ), let  $z_c$  be a variable and  $B = [b_0, b_1, \dots, b_{n-1}]$  be a set of variables such that  $\forall b_l \in B, D(b_l) = \{0, 1\}$ .  $\text{WEIGHTEDFOCUS}(X, y_c, \text{len}, k, z_c) \Leftrightarrow \text{FOCUS}(X, y_c, \text{len}, k) \wedge [\forall l, 0 \leq l < n, [(x_l \leq k) \wedge (b_l = 0)] \vee [(x_l > k) \wedge (b_l = 1)]] \wedge \sum_{l \in \{0, 1, \dots, n-1\}} b_l \leq z_c$ .*

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

## 4 Weighted Springy FOCUS

We consider a further generalization of the FOCUS constraint that combines 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.

**Definition 6.** *Let  $y_c$  and  $z_c$  be two variables and  $k, \text{len}, h$  be three integers, such that  $1 \leq \text{len} \leq |X|$  and  $0 < h < \text{len} - 1$ . An instantiation of  $X \cup \{y_c\} \cup z_c$  satisfies WEIGHTEDSPRINGYFOCUS( $X, y_c, \text{len}, h, k, z_c$ ) 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)  $\forall s_{i,j} \in S_X, j - i + 1 \leq \text{len}, x_i > k$  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.  $\sum_{s_{i,j} \in S} |s_{i,j}| = \sum_{s_{i,j} \in S} \text{cst}(s_{i,j}) + |P_k|$ . However,  $\text{cst}(s_{i,j})$  is the number of *undetermined* and *neutral* variables covered  $s_{i,j}$ ,  $\text{cst}(s_{i,j}) = \{p | x_p \in U_k \cup N_k, x_p \in s_{i,j}\}$  as we allow to cover up to  $h$  neutral variables.

The propagator is again based on a dynamic program that for each prefix of variables  $[x_0, x_1, \dots, x_j]$  and given cost  $c$  computes a cover  $S_{c,j}$  of minimum cardinality that covers all penalized variables in the prefix  $[x_0, x_1, \dots, x_j]$  and has

cost *exactly*  $c$ . We face the same problem of how to compare two sets  $S_{c,j}^1$  and  $S_{c,j}^2$  of minimum cardinality. The issue here is how to compare  $\text{last}(S_{c,j}^1)$  and  $\text{last}(S_{c,j}^2)$  if they cover a different number of neutral variables. Luckily, we can avoid this problem due to the following monotonicity property. If  $\text{last}(S_{c,j}^1)$  and  $\text{last}(S_{c,j}^2)$  are not equal to infinity then they both end at the same position  $j$ . Hence, if  $\text{last}(S_{c,j}^1) \leq \text{last}(S_{c,j}^2)$  then the number of neutral variables covered by  $\text{last}(S_{c,j}^1)$  is no larger than the number of neutral variables covered by  $\text{last}(S_{c,j}^2)$ . Therefore, we can define order on sets  $S_{c,j}$  as we did in Section 3 for WEIGHTEDFOCUS.

Our bounds disentailment detection algorithm for WEIGHTEDSPRINGYFOCUS mimics Algorithm 3. We omit the pseudocode due to space limitations but highlight two not-trivial differences between this algorithm and Algorithm 3. The first difference is that each cell in the dynamic programming table  $f_{c,j}$ ,  $c \in [0, z_c^U]$ ,  $j \in \{0, 1, \dots, n-1\}$ , where  $z_c^U = \max(z_c) - |P_k|$ , 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 3(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 3(c), the black arc). BC can enforced using two modifications of the corresponding algorithm for WEIGHTEDFOCUS (a proof is given in [?]).

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

WEIGHTEDSPRINGYFOCUS can be encoded using the cost-REGULAR constraint. The automaton needs 3 counters to compute  $\text{len}, y_c$  and  $h$ . Hence, the time complexity of this encoding is  $O(n^4)$ . This automaton is non-deterministic as on seeing  $v \leq k$ , it either covers the variable or interrupts the last sequence. Unfortunately the non-deterministic cost-REGULAR is not implemented in any constraint solver to our knowledge. In contrast, our algorithm takes just  $O(n^2)$  time. WEIGHTEDSPRINGYFOCUS can also be decomposed using the GCC constraint [?]. 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.  $D(S_i) = \{0, \dots, n + \max(y_c)\}$ ;  $E_i$  the end of the  $i$ th sub-sequence.  $D(E_i) = \{0, \dots, n + \max(y_c)\}$ ;  $T_j$  the index of the subsequence in  $S_X$  containing  $x_j$ .  $D(T_j) = \{0, \dots, \max(y_c)\}$ ;  $Z_j$  the index of the subsequence in  $S_X$  containing  $x_j$  s.t. the value of  $x_j$  is less than or equal to  $k$ .  $D(Z_j) = \{0, \dots, \max(y_c)\}$ ;  $\text{last}_c$  the cardinality of  $S_X$ .  $D(\text{last}_c) = \{0, \dots, \max(y_c)\}$ ;  $\text{Card}$ , a vector of  $\max(y_c)$  variables having  $\{0, \dots, h\}$  as domains.  $\text{WEIGHTEDSPRINGYFOCUS}(X, y_c, \text{len}, h, k, z_c) \Leftrightarrow$

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



Table 2: SLS with WEIGHTEDFOCUS and its decomposition.

		16.1			16.2			16.3		
	#n	#b	T	#n	#b	T	#n	#b	T	
F	50	0.9K	0	50	4.1K	2	47	18.1K	7	
D <sub>1</sub>	50	3.4K	1	49	8.1K	3	44	21.8K	8	

		20.1			20.2			20.3		
	#n	#b	T	#n	#b	T	#n	#b	T	
F	49	11.8K	7	45	24.9K	14	39	36.5K	23	
D <sub>1</sub>	43	30.8K	13	35	27.2K	12	29	29.6K	17	

$$last_c \leq y_c; \quad Gcc([T_0, \dots, T_{n-1}], \{0\}, [n - z_c]);$$

$$Gcc([Z_0, \dots, Z_{n-1}], \{1, \dots, max(y_c)\}, Card);$$

## 5 Experiments

We used the Choco-2.1.5 solver on an IntelXeon 2.27GHz for the first benchmarks and IntelXeon 3.20GHz for last ones, both under Linux. We compared the propagators (denoted by F) of WEIGHTEDFOCUS and WEIGHTEDSPRINGYFOCUS against two decompositions (denoted by D<sub>1</sub> and D<sub>2</sub>), using the same search strategies, on three different benchmarks. The first decomposition, restricted to WEIGHTEDFOCUS, is shown in proposition 4, while the second one is shown in Section 4. In the tables, we report for each set the total number of solved instances (#n), then we average both the number of backtracks (#b) and the resolution time (T) in seconds.

□ *Sports league scheduling (SLS)*. We extend a single round-robin problem with  $n = 2p$  teams. Each week each team plays a game either at home or away. Each team plays exactly once all the other teams during a season. We minimize the number of breaks (a break for one team is two consecutive home or two consecutive away games), while fixed weights in  $\{0, 1\}$  are assigned to all games: games with weight 1 are important for TV channels. The goal is to group consecutive 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. Table 2 shows results with 16 and 20 teams, on sets of 50 instances with 10 random important games and a limit of 400K backtracks.  $max(y_c) = 3$  and we search for one solution with  $h \leq 7$  (instances  $n-1$ ),  $h \leq 6$  ( $n-2$ ) and  $h \leq 5$  ( $n-3$ ). In our model, inverse-channeling and ALLDIFFERENT constraints with the strongest propagation level express that each team plays once against each other team. We assign first the sum of breaks by team, then the breaks and places.

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

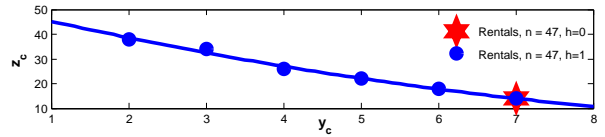


Figure 4: Pareto frontier for Scheduling with Rentals.

Table 3: Scheduling with Rentals.

		40			43			45		
h		#n	#b	T	#n	#b	T	#n	#b	T
0	F	20	349K	55.4	20	1M	192.2	20	1M	233.7
0	D <sub>1</sub>	20	529K	74.7	20	1M	251.2	20	1M	328.6
1	F	20	827M	120.4	20	2M	420.9	19	3M	545.9
2	F	20	826K	115.7	20	2M	427.3	19	3M	571.3

		47			50		
h		#n	#b	T	#n	#b	T
0	F	19	1M	354.5	18	2M	553.7
0	D <sub>1</sub>	18	2M	396.8	17	3M	660
1	F	16	4M	725.4	4	6M	984.5
2	F	15	4M	763.9	4	5M	944.8

size 20 (i.e.,  $len = 20$ ), 50% as a probability of posting an *Among* constraint for each  $(i, j)$  s.t.  $j \geq i + 5$  in the sequence. Each set of instances corresponds to a unique sequence size ( $\{40, 43, 45, 47, 50\}$ ) and 20 different seeds. We summarize these tests in table 3. Results with decomposition are very poor. We therefore consider only the propagator in this case.

□ *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. The full description and a CP model is in [?]. The main difference here is that we build a Pareto frontier over two cost variables. We generated 4 sets of instances distinguished by the numbers of chords ( $\{14, 16, 18, 20\}$ ). We fixed the length of the subsequences and the maximum notes for all the sets then change the seed for each instance.

Tables 2, 3 and 4 show that best results were obtained with our propagators (number of solved instances, average backtracks and CPU time over all the solved instances<sup>1</sup>). Figure 4 confirms the gain of flexibility illustrated by Figure 1 in Section 2: allowing  $h = 1$  variable with a low cost value into each sequence leads to new solutions, with significantly lower values for the target variable  $y_c$ .

## 6 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. Our experiments have demonstrated the effectiveness of the proposed new filtering algorithms.

<sup>1</sup>While the technique that solves the largest number of instances (and thus some harder ones) should be penalized.

Table 4: Sorting Chords

		14			16			18			20		
h		#n	#b	T	#n	#b	T	#n	#b	T	#n	#b	T
0	F	30	70K	2.8	30	865K	14.6	28	10M	182.9	<b>16</b>	14M	270.4
0	D <sub>1</sub>	30	94K	3.2	30	2M	41	28	12M	206.9	13	10M	206.8
0	D <sub>2</sub>	30	848K	34.9	24	3M	122.3	13	8M	285.6	7	902K	38.7
1	F	30	97K	3.5	<b>30</b>	1K	27.2	<b>28</b>	12K	214.2	<b>14</b>	13M	288.2
1	D <sub>2</sub>	30	851M	41.5	23	2M	116.3	11	5M	209.9	7	868K	41.5
2	F	30	97K	3.4	<b>30</b>	1M	25.9	<b>28</b>	13M	217.4	<b>13</b>	12M	245.5
2	D <sub>2</sub>	30	844K	40.9	24	3M	145.1	12	6K	251.6	7	867K	42.8