CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Revisiting two-sided stability constraints
Authors	Siala, Mohamed;O'Sullivan, Barry
Publication date	2016-05
Original Citation	Siala, M. and O'Sullivan, B. (2016) 'Revisiting Two-Sided Stability Constraints', in Quimper, C.-G. (ed.) Integration of AI and OR Techniques in Constraint Programming: 13th International Conference, CPAIOR 2016, Banff, AB, Canada, May 29 - June 1, 2016, Lecture Notes in Computer Science, vol. 9676, Cham: Springer International Publishing, pp. 342-357. doi: 10.1007/978-3-319-33954-2_25
Type of publication	Conference item
Link to publisher's version	10.1007/978-3-319-33954-2_25
Rights	© Springer International Publishing AG. The final publication is available at Springer via https://doi.org/10.1007/978-3-319-33954-2_25
Download date	2026-11-04 08:11:21
Item downloaded from	https://hdl.handle.net/10468/2480



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Revisiting Two-Sided Stability Constraints*

Mohamed Siala and Barry O'Sullivan

Insight Centre for Data Analytics
Department of Computer Science, University College Cork, Ireland
{mohamed.siala|barry.osullivan}@insight-centre.org

Abstract. We show that previous filtering propositions on two-sided stability problems do not enforce arc consistency (AC), however they maintain Bound($\mathcal{D}$) Consistency (BC(D)). We propose an optimal algorithm achieving BC(D) with $O(L)$ time complexity where L is the length of the preference lists. We also show an adaptation of this filtering approach to achieve AC. Next, we report the first polynomial time algorithm for solving the hospital/resident problem with forced and forbidden pairs. Furthermore, we show that the particular case of this problem for stable marriage can be solved in $O(n^2)$ which improves the previously best complexity by a factor of n^2. Finally, we present a comprehensive set of experiments to evaluate the filtering propositions.

1 Introduction

Many real world problems involve matching preferences between two sets of agents while respecting some stability criteria. For instance, in *College Admissions* one needs to assign students to colleges while respecting the students' preferences over colleges, the colleges' preferences over students, as well as college quotas [3]. Gale and Shapley introduced the first polynomial time algorithm for solving this problem in their seminal paper [3]. Since then a number of algorithms have been proposed for solving variants of these problems. Such ad-hoc methods are unlikely to be reusable if there are minor changes to the problem.

Constraint programming (CP) is a rich framework for modelling and solving many combinatorial problems. Expressing problems involving preferences in CP is extremely beneficial for tackling variants that involve side constraints. We consider the notion of two-sided stability as a global constraint. We first make the observation that the previous CP propositions on two-sided stability problems (such as [4, 9, 7, 10]) do not enforce Arc Consistency (AC), however they do maintain Bound($\mathcal{D}$) Consistency (BC(D)). We propose an incremental algorithm that achieves BC(D) with $O(L)$ time complexity where L is the length of the preference lists, thereby improving the previously best known complexity of $O(c \times L)$ (where c is the maximum quota). We also present, for the first time, an adaptation of the filtering to achieve AC on this global constraint with an additional cost of $n \times L$ (where n is the number of residents).

* This research has been funded by Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

Based on the BC(D) propagator, we show that the hospital/resident problem with forced and forbidden pairs can be solved in polynomial time. Furthermore, we show that the particular case of this problem for stable marriage can be solved in $O(n^2)$ which improves the previously best complexity by a factor of n^2. Finally, we present a set of experiments to evaluate the filtering efficiency on randomly generated instances. The experimental results show compelling evidence that AC does further prune the search space as compared with BC(D), however, it considerably slows down the exploration of the search space.

The remainder of the paper is organized as follows. In Section 2 we give the definitions and the notation used throughout the paper. In Section 3 we show that the level of filtering of previous CP approaches is only BC(D). Next, we show an optimal implementation of a BC(D) algorithm running in $O(L)$ time in Section 4. We also show how to use the same algorithm to achieve AC. In Section 5 we discuss the complexity of the hospital/resident problem with forced and forbidden pairs. Finally, we present the experimental results in Section 6.

2 Definitions and Related work

2.1 Constraint Programming

Let $\mathcal{X}$ be a set of integer variables. A *domain* for $\mathcal{X}$, denoted by $\mathcal{D}$, is a mapping from variables to finite sets of integers. For each variable x, we call $\mathcal{D}(x)$ the *domain of the variable* x. We use $\min(x)$ to denote the minimum value in $\mathcal{D}(x)$ and $\max(x)$ to denote the maximum value in $\mathcal{D}(x)$. Let $[x_1, \ldots, x_k]$ be a sequence of integer variables. A *constraint* C defined over $[x_1, \ldots, x_k]$ is a finite subset of $\mathbb{Z}^k$. The sequence $[x_1, \ldots, x_k]$ is the *scope* of C (denoted by $\mathcal{X}(C)$) and k is called the *arity* of C. A *support* for C in a domain $\mathcal{D}$ is a k-tuple τ such that $\tau \in C$ and $\tau[i] \in \mathcal{D}(x_i)$ for all $i \in [1, \ldots, k]$. The constraint C is *Arc-Consistent* (AC) in $\mathcal{D}$ iff $\forall i \in [1, \ldots, k]$, $\forall j \in \mathcal{D}(x_i)$, there exists a support τ for C in $\mathcal{D}$ such that $\tau[i] = j$. C is *Bound (D) Consistent* (*BC(D)*) in $\mathcal{D}$ iff $\forall i \in [1, \ldots, k]$, there exists two supports τ_1 and τ_2 for C in $\mathcal{D}$ such that $\tau_1[i] = \min(x_i)$ and $\tau_2[i] = \max(x_i)$ [1].

2.2 The Hospital/Resident Problem

Given a sequence $\mathcal{S}$ of distinct elements and $j \in \mathcal{S}$, we denote by $\mathcal{S}^{-1}[j]$ the index i such that $\mathcal{S}[i] = j$. We define a complete order $\underset{\mathcal{S}}{\prec\prec}$ on $\mathcal{S}$ as follows: $\forall k, l \in \mathcal{S},\ k \underset{\mathcal{S}}{\prec\prec} l$ iff $\mathcal{S}^{-1}[k] < \mathcal{S}^{-1}[l]$. We will also use the notation $l \underset{\mathcal{S}}{\succ\succ} k$ when $k \underset{\mathcal{S}}{\prec\prec} l$. In the context of preferences, $k \underset{\mathcal{S}}{\prec\prec} l$ (respectively $k \underset{\mathcal{S}}{\succ\succ} l$) can be understood as k is *better* (respectively *worse*) than l with respect to $\mathcal{S}$.

In the *Hospital/Resident* (*HR*) problem (called College Admissions in [3]), we are seeking the assignment of residents $r_1, \ldots, r_{n_R}$ to hospitals $h_1, \ldots, h_{n_H}$. Each hospital h_j has a capacity c_j as the maximum number of assigned residents. Each resident r_i has a sequence of integers $\mathcal{R}_i$ ranking some hospitals in a strictly

increasing order of preferences. That is, r_i prefers hospital h_k to hospital h_l iff $k \underset{\mathcal{R}_i}{\prec\prec} l$. Conversely, each hospital h_j is associated with a sequence of integers $\mathcal{H}_j$ ranking some residents in a strictly increasing order. We denote by $len^r{}_i$ the length of $\mathcal{R}_i$ and $len^h{}_j$ the length of $\mathcal{H}_i$.

Let $\mathcal{E} = \{(i,j) \mid i \in [1,n_R] \wedge j \in [1,n_H] \wedge i \in \mathcal{H}_j \wedge j \in \mathcal{R}_i\}$ the set of acceptable pairs. A *matching* $\mathcal{M}$ is a subset of $\mathcal{E}$ where $|\{j \mid (i,j) \in \mathcal{M}\}| \leq 1 \ \forall i \in [1,n_R]$ and $|\{i \mid (i,j) \in \mathcal{M}\}| \leq c_j, \ \forall j \in [1,n_H]$. A resident r_i is said to be *unassigned* in $\mathcal{M}$ iff $|\{j \mid (i,j) \in \mathcal{M}\}| = 0$. Similarly, a hospital h_j is *under-subscribed* in $\mathcal{M}$ iff $|\{i \mid (i,j) \in \mathcal{M}\}| < c_j$. A pair $(i,j) \in \mathcal{E} \setminus \mathcal{M}$ is said to be *blocking* $\mathcal{M}$ iff the following two conditions are true:

1. r_i is unassigned in $\mathcal{M}$ or $\exists k \in [1,n_H]$ such that $(i,k) \in \mathcal{M}$ and $j \underset{\mathcal{R}_i}{\prec\prec} k$
2. h_j is under-subscribed in $\mathcal{M}$ or $\exists l \in [1,n_R]$ such that $(l,j) \in \mathcal{M}$ and $i \underset{\mathcal{H}_j}{\prec\prec} l$

A matching $\mathcal{M}$ is said to be *stable* iff there is no blocking pair for $\mathcal{M}$.

The Hospital/Resident (*HR*) problem is to find a stable matching for a given instance. The stable marriage problem (*SM*) is a particular case of HR where $c_j = 1$ for all $j \in [1,n_H]$. We assume, without loss of generality in the remainder of the paper, that a resident r has a hospital h in its preference list iff h has r in its preference list. In this case, the length of the preference lists $L = \sum_{i=1}^{n_R} len^r{}_i = \sum_{j=1}^{n_H} len^r{}_j$.

Gale and Shapley proposed an $O(L)$ algorithm for solving the HR problem [3]. The algorithm, known as the *resident-oriented Gale/Shapley algorithm (RGS)* returns the unique matching where each resident is assigned to the best possible hospital that it can be assigned to in any stable matching. A similar algorithm for hospitals (i.e. *hospital-oriented Gale/Shapley algorithm (*HGS*)*) exists and has the same complexity $O(L)$. RGS and HGS operate by removing residents/hospitals from preference lists. The intersection of the reduced lists (returned by RGS and HGS) is called the GS-lists. The GS-lists are important since every stable matching is included in it [5].

Theorem 1. *From [5]*

1. *The number of assigned residents per hospital is the same in all stable matchings*
2. *If a resident r_i is unassigned in one stable matching then it is unassigned in all stable matchings.*
3. *If a hospital h_j is under-subscribed in one stable matching then it is assigned exactly the same residents in all stable matching*

We will use the following notation:

- $HUnder = \{j \mid h_j$ is under-subscribed in all stable matchings$\}$.
- $HUnder_j = \{i | r_i$ is assigned to h_j in all stable matchings$\}$ where $j \in HUnder$
- $HFull = [1,n_H] \setminus HUnder$.
- $RUnassigned = \{i \mid r_i$ is unassigned in all stable matchings$\}$.
- $RFree = \{i \mid r_i \notin RUnassigned$ and $\{j | i \in HUnder_j\} = \emptyset\}$.

2.3 Related Work in Constraint Programming

We first describe one of the CP models for the HR problem proposed in [7]. Each resident r_i is associated with an integer variable x_i where $\mathcal{D}(x_i) = [1, .., len^r{}_i] \cup \{n_H + 1\}$. Each hospital h_j is associated with $c_j + 1$ integer variables $y_{j,k}$ ($k \in [0..c_j]$) where $\mathcal{D}(y_{j,0}) = 0$ and $\mathcal{D}(y_{j,k}) = [k, len^h{}_j] \cup \{n + k\}$. Assigning x_i to $n_H + 1$ is understood as the resident r_i being unassigned. Assigning x_i to a value $a \in [1, len^r{}_i]$ is semantically equivalent to assigning r_i to its ath favourite hospital. Similarly, assigning $y_{j,k}$ to a value $b \in [1, len^h{}_j]$ means that the bth favourite resident to h_j is assigned to the kth position of h_j. If $y_{j,k}$ is assigned to $\{n + k\}$ then the k^{th} position for hospital h_j is not assigned to any resident. These variables are subject to the following constraints (in these constraints $p_{i,j}$ denotes the rank of hospital h_j in $\mathcal{R}_i$ and $q_{i,j}$ denotes the rank of the resident r_i in $\mathcal{H}_j$):

$$y_{j,k} < y_{j,k+1} \ (\forall j \in [1, n_H], \forall k \in [1, c_j - 1]) \tag{1}$$

$$y_{j,k} \geq q_{i,j} \implies x_i \leq p_{i,j} \ (\forall j \in [1, n_H], \forall k \in [1, c_j], \forall i \in \mathcal{H}_j) \tag{2}$$

$$x_i \neq p_{i,j} \implies y_{j,k} \neq q_{i,j} \ (\forall i \in [1, n_R], \forall j \in \mathcal{R}_i, \forall k \in [1, c_j]) \tag{3}$$

$$(x_i \geq p_{i,j} \wedge y_{j,k-1} < q_{i,j}) \implies y_{j,k} \leq q_{i,j} \ (\forall i \in [1, n_R], \forall j \in \mathcal{R}_i, \forall k \in [1, c_j]) \tag{4}$$

$$y_{j,c_j} < q_{i,j} \implies x_i \neq p_{i,j} \ (\forall j \in [1, n_H], \forall i \in \mathcal{H}_j) \tag{5}$$

We refer to this encoding as Γ. Enforcing AC on Γ yields to a domain that is equivalent to the GS-lists [7]. This property is important, however, it does not necessarily rule out all inconsistent values. The authors of [7] showed an efficient implementation of this encoding using one constraint. Their filtering runs in $O(c \times L)$ (where $c = \max\{c_j | j \in [1, n_H]\}$) and does not further prune the domains. In fact, in terms of the level of propagation, all previous work in the literature including SM [4, 9, 7, 10] focus on showing how their encodings maintains the GS-lists and never investigate the question of completing the filtering.

Note also that the notion of GS-lists is not well-defined during search. That is, for instance, when few residents are assigned/unassigned to some specific hospitals at a given node of the search tree.

3 Characterizing the Level of Consistency

We show in this section that the previous CP models are not complete and enforce only BC(D).

Example 1 (Counter-example). Consider the case where $n_R = n_H = 4, c_1 = c_2 = c_3 = c_4 = 1, \mathcal{R}_1 = [3, 2, 1], \mathcal{R}_2 = [4, 1, 3, 2], \mathcal{R}_3 = [2, 4, 3], \mathcal{R}_4 = [1, 3, 4], \mathcal{H}_1 = [1, 2, 4], \mathcal{H}_2 = [2, 1, 3], \mathcal{H}_3 = [3, 2, 4, 1], \mathcal{H}_4 = [4, 3, 2]$. The domain is initialised as follows: $\mathcal{D}(x_1) = \mathcal{D}(x_3) = \mathcal{D}(x_4) = \{1, 2, 3, 5\}, \mathcal{D}(x_2) = \{1, 2, 3, 4, 5\}, \mathcal{D}(y_{1,0}) = \mathcal{D}(y_{2,0}) = \mathcal{D}(y_{3,0}) = \mathcal{D}(y_{4,0}) = 0, \mathcal{D}(y_{1,1}) = \mathcal{D}(y_{2,1}) = \mathcal{D}(y_{4,1}) = \{1, 2, 3, 5\}$, and $\mathcal{D}(y_{3,1}) = \{1, 2, 3, 4, 5\}$. Constraint 2 with $y_{1,1} >= 1$ enforces $x_1 \leq 3$,

hence removing the value 5 from $\mathcal{D}(x_1)$. A similar propagation is performed on Constraint 2 with $y_{2,1} \geq 1, y_{3,1} \geq 1$, and $y_{4,1} \geq 1$ and the value 5 is removed from $\mathcal{D}(x_2), \mathcal{D}(x_3)$, and $\mathcal{D}(x_4)$. Now consider Constraint 4 with $x_1 \geq 1 \ \wedge \ y_{3,0} < 4$. This enforces $y_{3,1} \leq 4$. The same constraint is triggered with $x_2 \geq 1 \wedge y_{4,0} < 3$, $x_3 \geq 1 \wedge y_{2,0} < 3$, and $x_4 \geq 1 \wedge y_{1,0} < 3$. Therefore the value 5 is removed from $\mathcal{D}(y_{1,1}), \mathcal{D}(y_{2,1}), \mathcal{D}(y_{3,1})$, and $\mathcal{D}(y_{4,1})$. No more propagation is needed. However, assigning x_2 to 3 does not belong to any solution. □

Example 1 shows that AC on Γ is not sufficient to provide complete filtering. Note that since all capacities $c_j = 1$ this example confirms the property even for the particular case of stable matching.

In the rest of the paper we use 2-SidedStability$(\mathcal{X}, \mathcal{A}, \mathcal{B}, \mathcal{C})$ to denote the global constraint modelling 2-sided stability. More precisely, for a given HR problem:

- $\mathcal{X}$ is the set of variables $x_1, \ldots, x_{n_R}$ defined the same way in Γ,
- $\mathcal{A} = \{\mathcal{R}_1, \ldots, \mathcal{R}_{n_R}\}$
- $\mathcal{B} = \{\mathcal{H}_1, \ldots, \mathcal{H}_{n_H}\}$
- $\mathcal{C} = \{c_1, \ldots, c_{n_H}\}$

We show that AC on Γ enforces BC(D) on any domain $\mathcal{D}$.

Lemma 1. *If Γ is AC then*

1. $\forall i \in RUnassigned$, $\mathcal{D}(x_i) = \{n_H + 1\}$.
2. $\forall j \in HUnder$, $\forall i \in HUnder_j$, $\mathcal{D}(x_i) = \{k\}$ *such that* $k = {\mathcal{R}_i}^{-1}[j]$.
3. $\forall j \in HUnder$, $\forall k \in [1, |HUnder_j|]$, $\mathcal{D}(y_{j,k}) = \{a_k\}$ *such that a_k is the kth favourite resident to h_j whose index is in $HUnder_j$.*
4. $\forall j \in HUnder$, $\forall k > |HUnder_j|$, $\mathcal{D}(y_{j,k}) = \{n + k\}$.
5. $\forall j \in HFull$, $\forall i \in [1, n_R]$, *if* $\exists k \in \mathcal{D}(x_i)$, *such that* $j = \mathcal{R}_i[k]$, *then* $i \in RFree$.
6. $\forall i \in RFree$, $\forall j \in [1, n_H]$ *if* $\exists k \in \mathcal{D}(x_i)$, *such that* $j = \mathcal{R}_i[k]$, *then* $j \in HFull$.
7. $|RFree| = \sum_{j \in HFull} c_j$.

Proof. The lemma is a direct consequence of Theorem 1 and the fact that AC on the initial domain is a superset of any domain returned by AC in the search tree. Recall that AC on the initial domain is equivalent to the GS-lists. □

Lemma 2. *If Γ is AC then for all $i \in [1, n_R]$, $1 \leq k < \max(x_i)$, and $h = \mathcal{R}_i[k]$, we have $h \in HFull$.*

Proof. Consider the domain $\mathcal{D}^*$ obtained after enforcing AC on the initial domain. We know that this domain corresponds to the GS-lists. Therefore, assigning all variables to their maximum in $\mathcal{D}^*$ is a support (i.e. the hospital-oriented stable matching). Hence for all $1 \leq k < \max(\mathcal{D}^*(x_i))$, and $h = \mathcal{R}_i[k]$, $h \in HFull$ (from the definition of stability). Therefore, if Γ is AC on any arbitrary domain, then we know that $\max(x_i) \leq m$, and consequently for all $1 \leq k < \max(x_i)$, and $h = \mathcal{R}_i[k]$, we have $h \in HFull$. □

Lemma 3. *If Γ is AC then $\forall j \in HFull$, $|\{i \mid \mathcal{R}_i[\min(x_i)] == j\}| = c_j$.*

Proof. Let $\Phi_j = \{i \mid \mathcal{R}_i[\min(x_i)] == j\}$. We first show that $|\Phi_j| \leq c_j$. Suppose by contradiction that $\exists j \in \mathit{HFull}$, $|\Phi_j| > c_j$. For all $k \in [1, |\Phi_j|]$, we define r_{a_k} to be the kth favourite resident to h_j whose index is in Φ_j (i.e. $a_k \in \Phi_j$).

We show by induction that $\forall k \in [1, c_j], \max(y_{j,k}) \leq {\mathcal{H}_j}^{-1}[x_{a_k}]$. For $k = 1$, since Constraint 4 of Γ is AC, and $y_{j,0} = 0 < {\mathcal{H}_j}^{-1}[x_{a_1}]$ then $\max(y_{j,1}) \leq {\mathcal{H}_j}^{-1}[x_{a_1}]$. Suppose that the property holds for $k \in [1, c_j - 1]$. We have $\max(y_{j,k}) \leq {\mathcal{H}_j}^{-1}[x_{a_k}]$. Therefore, Constraint 4 of Γ enforces $\max(y_{j,k+1}) \leq {\mathcal{H}_j}^{-1}[x_{a_{k+1}}]$ since ${\mathcal{H}_j}^{-1}[x_{a_k}] < {\mathcal{H}_j}^{-1}[x_{a_{k+1}}]$.

Consider now $k \in [c_j + 1, |\Phi_j|]$. We have $\max(y_{j,c_j}) < {\mathcal{H}_j}^{-1}[x_{a_k}]$ (since ${\mathcal{H}_j}^{-1}[x_{a_{c_j}}] < {\mathcal{H}_j}^{-1}[x_{a_k}]$). Therefore Constraint 5 of Γ removes ${\mathcal{R}_{a_k}}^{-1}[j]$ from $\mathcal{D}(x_{a_k})$ which contradicts the fact that $\mathcal{R}_{a_k}[\min(x_{a_k})] = j$. Hence $|\Phi_j| \leq c_j$.

Using Properties 5, 6, and 7 of Lemma 1 we obtain

$$\sum_{j \in \mathit{HFull}} |\Phi_j| = \sum_{j \in \mathit{HFull}} c_j.$$

Therefore, $|\Phi_j| = c_j$. □

Lemma 4. *If Γ is AC then $\forall j \in \mathit{HFull}$, $|\{i \mid \mathcal{R}_i[\max(x_i)] == j\}| = c_j$.*

Proof. We show first that $\forall j \in \mathit{HFull}$, $\forall k \in [1, c_j]$, if $\mathcal{H}_j[\min(y_{j,k})] = i$, then $\mathcal{R}_i[\max(x_i)] = j$. Suppose by contradiction that there exists j, k, i such that $\mathcal{H}_j[\min(y_{j,k})] = i$ and $\mathcal{R}_i[\max(x_i)] \neq j$. Then Constraint 2 enforces $\max(x_i) \leq {\mathcal{R}_i}^{-1}[j]$. Therefore we have $\max(x_i) < {\mathcal{R}_i}^{-1}[j]$. Thus, Constraint 3 removes ${\mathcal{H}_j}^{-1}[i]$ from $\mathcal{D}(y_{j,k})$ which contradicts the hypothesis. Hence we have $\forall j \in \mathit{HFull}$, $\forall k \in [1, c_j]$, if $\mathcal{H}_j[\min(y_{j,k})] = i$, then $\mathcal{R}_i[\max(x_i)] = j$.

Observe that propagating Constraint 1 enforces $\min(y_{j,1})$, $\min(y_{j,2})$, …, $\min(y_{j,c_j})$ to have different values. Therefore $\forall j \in \mathit{HFull}$,$|\{\min(y_{j,k}) \mid k \in [1, c_j]\}| = c_j$. Thus $|\{i \mid \mathcal{R}_i[\max(x_i)] == j\}| \geq c_j$.

Since $\mathcal{R}_i[\max(x_i)] == j$ is true only when $i \in \mathit{RFree}$, and for all $i \in \mathit{RFree}$, $\mathcal{R}_i[\max(x_i)] \in \mathit{HFull}$ then $\sum_{j \in \mathit{HFull}} |\{i \mid \mathcal{R}_i[\max(x_i)] == j\}| = |\mathit{RFree}| = \sum_{j \in \mathit{HFull}} c_j$. Therefore, $\forall j \in \mathit{HFull}$, $|\{i \mid \mathcal{R}_i[\max(x_i)] == j\}| = c_j$. □

We now introduce a sufficient and necessary condition for stability in Theorem 2.

Theorem 2. 2-SidedStability$(\mathcal{X}, \mathcal{A}, \mathcal{B}, \mathcal{C})$ *is satisfiable iff*

$$\forall\ 1 \leq j \leq n_H,\ \sum_{i=1}^{n_R} (\mathcal{R}_i[x_i] == j) \leq c_j\ \wedge$$

$$\forall\ 1 \leq i \leq n_R,\ \forall\ 1 \leq j \leq n_H + 1, x_i = j\ \implies\ \forall k \in [1, j[,\ \textit{if } h = \mathcal{R}_i[k],\ \textit{then}$$

$$\sum_{m=1}^{n_R} (\mathcal{R}_m[x_m] == h) = c_h \wedge \forall l \underset{\mathcal{H}_h}{\succ\!\succ} i,\ \mathcal{R}_l[x_l] \neq h.$$

Proof. ($\Rightarrow$) Let $\mathcal{M}$ be a stable matching and suppose that the variables are assigned accordingly. Clearly, by construction, $\forall\ 1 \leq j \leq n_H$, $\sum_{i=1}^{n_R} (\mathcal{R}_i[x_i] == j) \leq c_j$. Let x_i be assigned to j, $1 \leq k < j$, and $h = \mathcal{R}_i[k]$. We show that $\sum_{m=1}^{n_R} (\mathcal{R}_m[x_m] == h) = c_h \wedge \forall l \underset{\mathcal{H}_h}{\succ\!\succ} i, \mathcal{R}_l[x_l] \neq h$.

If we suppose by contradiction that $\sum_{m=1}^{n_R}(\mathcal{R}_m[x_m] == h) \neq c_h \vee \exists l \underset{\mathcal{H}_h}{\succ\succ} i, \mathcal{R}_l[x_l] = h$, then, $\mathcal{H}_h$ is under subscribed or $\exists (l,h) \in \mathcal{M}$ and $i \underset{\mathcal{H}_h}{\prec\prec} l$. Therefore, (i,h) is blocking $\mathcal{M}$. Hence the contradiction.
($\Leftarrow$) Consider an assignment of the the variables $x_1, .., x_n$ satisfying the property. We show that the corresponding matching $\mathcal{M}$ is stable. If $\mathcal{M}$ is not stable then there exists a blocking pair (a,b). There are two cases to consider:

- r_a is unassigned in $\mathcal{M}$: In this case $x_a = n_H + 1$, hence $\forall h \in [1, n_H], \sum_{l=1}^{l=n_R}(\mathcal{R}_l[x_l] == h) = c_h \wedge \forall l \underset{\mathcal{H}_h}{\succ\succ} a, \mathcal{R}_l[x_l] \neq h$. Therefore h_b cannot be under-subscribed and there does not exist $l \in [1, n_R]$ such that $(l,b) \in \mathcal{M}$ and $a \underset{\mathcal{H}_b}{\prec\prec} l$.
- $\exists k \in [1, n_H]$ such that $(a,k) \in \mathcal{M}$ and $b \underset{\mathcal{R}_a}{\prec\prec} k$: In this case $x_a = e$ where $e = {\mathcal{R}_a}^{-1}[k]$, hence for all $w \underset{\mathcal{H}_a}{\prec\prec} k, \sum_{m=1}^{n_R}(\mathcal{R}_m[x_m] == w) = c_w \wedge \forall l \underset{\mathcal{H}_w}{\succ\succ} a, \mathcal{R}_l[x_l] \neq w$. Since $b \underset{\mathcal{R}_a}{\prec\prec} k$, then h_b cannot be under-subscribed and there does not exist $l \in [1, n_R]$ such that $(l,b) \in \mathcal{M}$ and $a \underset{\mathcal{H}_b}{\prec\prec} l$.

Therefore $\mathcal{M}$ is stable. □

Lemma 5. *If Γ is AC then assigning all variables to their minimum value is solution.*

Proof. We use Theorem 2 to prove that assigning all variables to their minimum satisfies the constraint. We already know that $\forall j \in [1, n_H], |\{i \mid \mathcal{R}_i[\min(x_i)] == j\}| \leq c_j$ by Lemma 1 and Lemma 3. Let $1 \leq i \leq n_R$, $j = \min(x_i)$, $k \in [1, j[$, $h = \mathcal{R}_i[k]$. We show that $\sum_{m=1}^{n_R}(\mathcal{R}_m[\min(x_m)] == h) = c_h \wedge \forall l \underset{\mathcal{H}_h}{\succ\succ} i, \mathcal{R}_l[\min(x_l)] \neq h$. Note first that $h \in \mathit{HFull}$ (Lemma 2). Therefore, by using Lemma 3 we obtain $\sum_{m=1}^{n_R}(\mathcal{R}_m[\min(x_m)] == h) = c_h$. Recall that $min(x_i) > {\mathcal{R}_i}^{-1}[h]$. Therefore, Constraint 2 of Γ enforces $\max(y_{h,k}) < {\mathcal{H}_h}^{-1}[i]$ for all $k \in [1, c_h]$. Thus, Constraint 5 removes ${\mathcal{R}_l}^{-1}[h]$ from $\mathcal{D}(x_l)$ for all $l \underset{\mathcal{H}_h}{\succ\succ} i$. □

Lemma 6. *If Γ is AC then assigning all variables to their maximum value is solution.*

Proof. Here again we use Theorem 2 to prove the result. First observe that $\forall j \in [1, n_H]$, $|\{i \mid \mathcal{R}_i[\max(x_i)] == j\}| \leq c_j$ by Lemma 1 and Lemma 4. Let $1 \leq i \leq n_R$, $j = \max(x_i)$, $k < j$, and $h = \mathcal{R}_i[k]$, we show that $\sum_{m=1}^{n_R}(\mathcal{R}_m[\max(x_m)] == h) = c_h \wedge \forall l \underset{\mathcal{H}_h}{\succ\succ} i, \mathcal{R}_l[\max(x_l)] \neq h$. Observe first that $h \in \mathit{HFull}$ (Lemma 2). Therefore, by using Lemma 4 we have $\sum_{m=1}^{n_R}(\mathcal{R}_m[\max(x_m)] == h) = c_h$. Next, we show that for any l such that $\mathcal{R}_l[\max(x_l)] = h$, then $\exists k \in [1, c_h]$ such that $\min(y_{h,k}) = {\mathcal{H}_h}^{-1}[l]$. In fact, if it's not the case, then by the proof of Lemma 4, we know that if $\min(y_{h,k}) = {\mathcal{H}_h}^{-1}[a]$ (for any $a \in [1, n_R]$), then $\mathcal{R}_a[\max(x_a)] = h$ and in this case we have $\sum_{m=1}^{n_R}(\mathcal{R}_m[\max(x_m)] == h) > c_h$ which contradicts Lemma 4 because $h \in \mathit{HFull}$.

Suppose now by contradiction that $\exists l \underset{\mathcal{H}_h}{\succ\!\succ} i, \mathcal{R}_l[\max(x_l)] = h$ and consider k such that ${\mathcal{H}_h}^{-1}[\min(y_{h,k})] = l$. Since $l \underset{\mathcal{H}_h}{\succ\!\succ} i$ then $\min(y_{h,k}) \geq {\mathcal{H}_h}^{-1}[i]$. Therefore, Constraint 2 enforces $\max(x_i) \leq {\mathcal{R}_i}^{-1}[h]$ which is false since $h \underset{\mathcal{R}_i}{\prec\!\prec} \mathcal{R}_i[\max(x_i)]$. Therefore we have $\forall l \underset{\mathcal{H}_h}{\succ\!\succ} i, \mathcal{R}_l[\max(x_l)] \neq h$. □

The following Theorem is an immediate consequence of Lemma 5 and Lemma 6.

Theorem 3. *Enforcing AC on Γ makes* 2-SidedStability$(\mathcal{X}, \mathcal{A}, \mathcal{B}, \mathcal{C})$ *Bound($\mathcal{D}$) consistent.*

4 Revisiting Bound($\mathcal{D}$) Consistency

We assume that a preprocessing step is performed where the GS-lists are computed and that the domain is updated accordingly. We suppose without loss of generality that $\exists n \in [1, n_R]$ such that $\forall i \in [1, n]$, $i \in \mathit{RFree}$ and $|\mathit{RFree}| = n$. Note that $\forall i > n$, $|\mathcal{D}(x_i)| = 1$ after the preprocessing step. Therefore, we shall assume that this holds for the rest of the section and we will focus only on $[x_1, \ldots, x_n]$. We show that BC(D) on 2-SidedStability$(\mathcal{X}, \mathcal{A}, \mathcal{B}, \mathcal{C})$ can be implemented with $O(L)$ time complexity down a branch of the search tree which improves the previous complexity $O(c \times L)$ (where $c = \max\{c_j | j \in [1, n_H]\}$). Next we show an adaptation of the filtering to achieve AC on this constraint.

4.1 Bound($\mathcal{D}$) Consistency

Our revision of BC(D) for this constraint is based essentially on Theorem 4.

Theorem 4. 2-SidedStability$(\mathcal{X}, \mathcal{A}, \mathcal{B}, \mathcal{C})$ *is BC(D) iff assigning every variable to its maximum is a solution and assigning every variable to its minimum is a solution.*

Proof. ($\Rightarrow$) Let $\mathcal{D}$ be a domain where 2-SidedStability$(\mathcal{X}, \mathcal{A}, \mathcal{B}, \mathcal{C})$ is BC(D). Consider the encoding Γ on a domain $\mathcal{D}^*$ where $\mathcal{D}^*(x_i) = \mathcal{D}(x_i)$ for all $x_i \in \mathcal{X}$ and $\mathcal{D}^*(y_{j,k})$ equals to the initial domain detailed in Section 2.3 for all $j \in [1, n_H]$ and $k \in [0, c_j]$. Let $\mathcal{D}'$ be the domain obtained after enforcing AC on Γ. We know that $\mathcal{D}'$ cannot be empty (i.e., failure) since otherwise it will contradict the fact that 2-SidedStability$(\mathcal{X}, \mathcal{A}, \mathcal{B}, \mathcal{C})$ is BC(D). Moreover, no lower/upper bound can change after AC on Γ since every lower/upper bound has a support. Therefore, by using Lemmas 5 and 6, we know that assigning every variable to its minimum (respectively maximum) is a solution.

($\Leftarrow$) Straightforward. □

Theorem 4 shows that in order to maintain BC(D), it is sufficient to make sure that two specific solutions exist: one by assigning all variables to their minimum, and the other to their maximum. In the following, we show that one can maintain this property in $O(L)$ time down a branch of the search tree using an incremental algorithm.

Given a domain $\mathcal{D}$, we define for each hospital h in *HFull* the following:

Algorithm 1: BC(D)

1 **while** $\exists < i, j > \in New_{lb}$ **do**
  UpdateLB$(i, j, \min(x_i) - 1, New_{lb})$;
  Apply(i, New_{lb}) ;
2 **while** $\exists < i, j > \in New_{ub}$ **do**
  UpdateUB(i, j, New_{ub}) ;

- $MIN_h = \{k \mid \mathcal{R}_{\mathcal{H}_h[k]}[\min(x_{\mathcal{H}_h[k]})] = h\}$
- $MAX_h = \{k \mid \mathcal{R}_{\mathcal{H}_h[k]}[\max(x_{\mathcal{H}_h[k]})] = h\}$
- $maxofMAX_h = \max\{l \mid l \in MAX_h\}$.

For any $k \in MIN_h$ (respectively $k \in MAX_h$), if r is the correspondent resident (i.e., $r = \mathcal{H}_h[k]$), then h is the hospital of index $\min(x_r)$ (respectively $\max(x_r)$) in $\mathcal{R}_r$.

We also define $lastLeft_h$ for every hospital $h \in HFull$ as follows: $lastLeft_h = max\{k \mid k \in [1, len^h{}_h] \wedge \mathcal{R}_{\mathcal{H}_h[k]}{}^{-1}[h] \in \mathcal{D}(x_{\mathcal{H}_h[k]})\}$. That is $lastLeft_h$ is the last index in the list of $\mathcal{H}_h$ where the corresponding resident still has the rank of h in its domain.

We suppose that MIN_h, MAX_h, $maxofMAX_h$, and $lastLeft_h$ are implemented as "reversible" data structures (i.e. their values are restored whenever the solver backtracks).

Let $\mathcal{D}_{BC(D)}$ be a domain that is BC(D) for 2-SidedStability$(\mathcal{X}, \mathcal{A}, \mathcal{B}, \mathcal{C})$. Let New_{lb} (respectively New_{ub}) be a set of pairs such that $< i, j > \in New_{lb}$ (respectively $< i, j > \in New_{ub}$) iff the lower (respectively upper) bound j has been removed from $\mathcal{D}_{BC(D)}(x_i)$. We show that Algorithm 1 maintains BC(D).

Take the case where only one variable x_i has a new lower bound $\min(x_i)$ and j was the previous lower bound. To maintain BC(D), we need to compute the new domain where assigning all variables to their minimum/maximum value is a solution. We therefore need to maintain $\forall\ 1 \leq j \leq n_H,\ \sum_{i=1}^{n_R}(\mathcal{R}_i[x_i] == j) \leq c_j$, and $\forall a \in [1, n], \forall h \underset{\mathcal{R}_a}{\prec\prec} \mathcal{R}_a[\min(x_a)]$, $\forall l \underset{\mathcal{H}_h}{\succ\succ} a, \mathcal{R}_l[\min(x_l)] \neq h$. In other words, $\forall a \in [1, n], \forall v < \min(x_a), \forall k > idx, \min(x_r) \neq \mathcal{R}_r{}^{-1}[h]$ where $h = \mathcal{R}_i[v]$, $idx = \mathcal{H}_h{}^{-1}[i]$, and $r = \mathcal{H}_h[k]$.

Consider first the variable x_i. Since $\mathcal{D}_{BC(D)}$ is BC(D), then the property already holds for all $v < j$. Let $[a, b] = [j, \min(x_i) - 1]$. The property must be enforced for all $v \in [a, b]$. This is precisely what Algorithm 2 does. Let $h \leftarrow \mathcal{R}_i[v]$. Observe that $\forall k > lastLeft_h$, if $r = \mathcal{H}_h[k]$ then $\mathcal{R}_r{}^{-1}[h] \notin \mathcal{D}(x_r)$ (from the definition of $lastLeft_h$). Therefore, one needs only to enforce the property for $k \in [idx, lastLeft_h]$ (Line 1) where $idx = \mathcal{H}_h{}^{-1}[i]$ and perform the filtering in Line 5. Note that this value removal might change the lower (respectively upper) bound for a given x_r. In this case, the pair $\langle r, \min(x_r)\rangle$ (respectively $\langle r, \max(x_r)\rangle$) is added to New_{lb} (respectively New_{ub}) in Line 3 (respectively Line 4). The case where $k \in MIN_h$ is handled at Line 2 by removing k from MIN_h.

Algorithm 2: UpdateLB(i, a, b, New_{lb})

```
   for v ∈ [a, b] do
       h ← R_i[v] ;
       idx ← H_h^{-1}[i] ;
1      for k ∈ [idx, lastLeft_h] do
           r ← H_h[k] ;
           if k ∈ MIN_h then
2              MIN_h ← MIN_h \ {k} ;
           if R_r^{-1}[h] == min(x_r) then
3              New_lb ← New_lb ∪ {⟨r, min(x_r)⟩} ;
           if R_r^{-1}[h] == max(x_r) then
4              New_ub ← New_ub ∪ {⟨r, max(x_r)⟩} ;
5          D(x_r) ← D(x_r) \ {R_r^{-1}[h]} ;
       if lastLeft_h > idx − 1 then
6          lastLeft_h ← idx − 1 ;
```

Now once the call to UpdateLB$(i, j, \min(x_i) - 1, New_{lb})$ ends, we have to make sure that it is actually possible to assign x_i to its new minimum. This is performed by calling Algorithm 3 Apply(i, New_{lb}). More precisely, let $h = \mathcal{R}_i[\min(x_i)]$. Obviously if ${\mathcal{H}_h}^{-1}[i] > lastLeft_h$ then x_i cannot be assigned to its minimum (Lines 9 and 10). Otherwise, ${\mathcal{H}_h}^{-1}[i]$ is added to MIN_h (Line 1). Suppose now that MIN_h has more than c_h elements. We can easily show that this happens only if $|MIN_h| = c_h + 1$. In this case, we can see that the resident associated with the maximum index in MIN_h cannot be assigned to hospital h anymore. The maximum is computed by looking for the first index less than or equal to MIN_h (Line 3). Lines 4, 5, 6, 7, and 8 handle the fact that the corresponding resident cannot be assigned to hospital h.

At the end of the first loop in Algorithm 1, we may argue that $|MIN_h| = c_h$. Therefore $\forall\ 1 \leq j \leq n_H,\ \sum_{i=1}^{n_R}(\mathcal{R}_i[x_i] == j) \leq c_j$, and $\forall a \in [1, n], \forall h \underset{\mathcal{R}_a}{\prec\prec} \mathcal{R}_a[\min(x_a)]$, $\forall l \underset{\mathcal{H}_h}{\succ\succ} a, \mathcal{R}_l[\min(x_l)] \neq h$. Thus assigning all variable to their minimum is a solution.

Consider now the case where only one variable x_i has a new upper bound $\max(x_i)$ and j was the previous upper bound. We use the following lemma to to compute the new domain.

Lemma 7. *Assigning all variables to their maximum is a solution iff* $\forall\ h \in HFull, |MAX_h| = c_h$, *and* $\forall\ i \leq maxofMAX_h$, *let* $r = \mathcal{H}_h[i]$, *and* $l = {\mathcal{R}_r}^{-1}[h]$, *then* $i \notin MAX_h \implies \max(x_r) < l$.

Proof. ($\Rightarrow$) Let M be the matching where each resident is assigned to the hospital corresponding to the maximum value in its domain. Let $h \in HFull$. We know that $|MAX_h| = c_h$ since M is stable. Consider now $i \leq maxofMAX_h$, such that $i \notin MAX_h$. Let $r = \mathcal{H}_h[i]$, and $l = {\mathcal{R}_r}^{-1}[h]$. Observe first that $\max(x_r) \neq l$

Algorithm 3: Apply(i, New_{lb})

```
   j ← min(x_i) ;
   h ← R_i[j] ;
   if H_h^{-1}[i] ≤ lastLeft_h then
1      MIN_h ← MIN_h ∪ {H_h^{-1}[i]} ;
       if |MIN_h| = c_h + 1 then
2          max ← lastLeft_h ;
           max_found = false ;
3          while not(max_found) do
               if max ∈ MIN_h then
                   max_found = true ;
               else
                   max ← max − 1
4          MIN_h ← MIN_h \ {max} ;
5          l = H_h[max] ;
6          D(x_l) ← D(x_l) \ {H_l^{-1}[h]} ;
7          New_lb ← New_lb ∪ {⟨l, min(x_l)⟩} ;
8          lastLeft_h ← max
   else
9      D(x_i) ← D(x_i) \ {min(x_i)} ;
10     New_lb ← New_lb ∪ ⟨i, min(x_i)⟩ ;
```

(otherwise $i \in MAX_h$). Next, one can easily show that if $\max(x_r) > l$ then the pair (r, h) blocks M. Therefore, $\max(x_r) < l$.

($\Leftarrow$) We use Theorem 2 to show that assigning all variables to their maximum is a solution. We already have $\forall\ 1 \leq j \leq n_H,\ \sum_{i=1}^{n_R}(\mathcal{R}_i[\max(x_i)] == j) \leq c_j$. We show that $\forall\ 1 \leq i \leq n_R,\ \forall\ 1 \leq j \leq n_H + 1, \max(x_i) = j \implies \forall k \in [1, j[$, if $h = \mathcal{R}_i[k]$, then $\sum_{m=1}^{n_R}(\mathcal{R}_m[\max(x_m)] == h) = c_h \wedge \forall l \underset{\mathcal{H}_h}{\succ\!\succ} i, \mathcal{R}_l[\max(x_l)] \neq h$. Note first that the property is true for all $i \in [n+1, n_R]$. Let $i \in [1, n]$ $j = \max(x_i)$, and $h \underset{\mathcal{R}_i}{\prec\!\prec} \mathcal{R}_i[j]$ (note that $j \neq n_H + 1$). We have necessarily $h \in \mathit{HFull}$ (Lemma 2), and therefore $\sum_{m=1}^{n_R}(\mathcal{R}_m[\max(x_m)] == h) = c_h$ since $|MAX_h| = c_h$. Let $l \underset{\mathcal{H}_h}{\succ\!\succ} i$. We show that $\mathcal{R}_l[\max(x_l)] \neq h$. Suppose by contradiction that $\mathcal{R}_l[\max(x_l)] = h$, and let $m = \mathcal{H}_h^{-1}[l]$. We have $m \in MAX_h$, hence $m \leq \max\{MAX_h\}$. Therefore, $\mathcal{H}_h^{-1}[i] < \max\{MAX_h\}$ because $l \underset{\mathcal{H}_h}{\succ\!\succ} i$. Since $\mathcal{H}_h^{-1}[i] \notin MAX_h$, then $\max(x_i) < \mathcal{R}_i^{-1}[h]$ which contradicts the hypothesis that $h \underset{\mathcal{R}_i}{\prec\!\prec} \mathcal{R}_i[j]$. □

We maintain the property in Lemma 7 by calling Algorithm 4, UpdateUB(i, j, New_{ub}). Recall that, in this case, the current upper bound of x_i is strictly less than j.

Let $h = \mathcal{R}_i[j]$ and $idx = \mathcal{H}_h^{-1}[i]$. There are two cases to consider. First if $idx \in MAX_h$, we know that MAX_h has to be of cardinality c_h. Therefore, a new index needs to be added to MAX_h. From Lemma 7 we can argue that the

Algorithm 4: UpdateUB(i, j, New_{ub})

```
   h ← R_i[j] ;
   idx ← H_h^{-1}[i] ;
   if idx ∈ MAX_h then
1      MAX_h ← MAX_h \ {idx} ;
       new_resident ← false ;
       max = maxofMAX_h ;
2      do
           max ← max + 1 ;
           r ← H_h[max] ;
           if R_r^{-1}[h] ≤ max(x_r) then
               new_resident ← true ;
       while not new_resident;
3      MAX_h ← MAX_h ∪ {max} ;
4      maxofMAX_h ← max ;
       rankOfh = R_r^{-1}[h] ;
       if rankOfh < max(x_r) then
5          New_ub ← New_ub ∪ {⟨r, max(x_r)⟩} ;
           % make a new upper bound for x_r ;
6          D(x_r) ← D(x_r) ∩ ]−∞, rankOfh] ;
   else
       if j − 1 > max(x_i) then
           New_ub ← New_ub ∪ {⟨i, j − 1⟩} ;
```

new index cannot correspond to a value less than or equal to $maxofMAX_h$. Therefore, we start looking for a new index in the main loop (Line 2) starting from $maxofMAX_h + 1$. The loop ends when we find a replacement for r_i. The new index corresponds to a resident r such that ${\mathcal{R}_r}^{-1}[h] \leq \max(x_r)$. The set MAX_h is updated accordingly in Lines 1 and 3. The upper bound of x_r is changed if h is better than the hospital corresponding to $\max(x_r)$ (Line 6). In this case, $\langle r, h\rangle$ is added to New_{ub} in Line 5.

Second, in the case where $idx \notin MAX_h$, we just need to make sure that if $j-1$ is not the current upper bound of $\max(x_i)$, then we need to simulate the case where $j-1$ was an upper bound for x_r and has been removed.

To maintain BC(D) in Algorithm 1, we loop over all the lower/upper bound changes and call the appropriate algorithms. The new domain is BC(D) as an immediate consequence of Theorems 4, and 2, and Lemma 7.

Complexity. We discuss now the complexity of this incremental algorithm. We assume that all set operations are implemented in constant time.

Observe that down a branch of the search tree the value of $lastLeft_h$ can always decrement (Line 6 in Algorithm 2 and Line 8 in Algorithm 3). Now consider one iteration of the main loop in Algorithm 2. The number of operations is bounded by $O(|idx - lastLeft_h|)$. Similarly, one call to Algorithm 3 is bounded

by $O(|m - m'|)$ where m (respectively m') is the value of $lastLeft_h$ at Line 2 (respectively 8). Since $lastLeft_h$ can only decrement, then the time complexity of all lower bound operations down a branch is $O(len^h{}_h)$ for any hospital h. Therefore the complexity for lower bound operations is $O(L)$.

Consider now upper bound operations. Each call to Algorithm 4 is bounded by $O(|m' - m|)$ where m and m' are the values of $maxofMAX_h$ at Lines 1 and 4 respectively. And since $maxofMAX_h$ can only increment down a branch of the search tree, then the number of upper bound operations for any hospital h is $O(len^h{}_h)$. Therefore the total upper bound operations is $O(L)$. Hence the complexity of enforcing BC(D) is $O(L)$ down a branch of the search tree.

4.2 Arc Consistency

Assume that 2-SidedStability$(\mathcal{X}, \mathcal{A}, \mathcal{B}, \mathcal{C})$ is BC(D). We use a straightforward way to enforce AC based on the BC(D) propagator. For every variable x_i, we remove its upper bound u then we enforce BC(D). Let u' the new upper bound for x_i. Clearly any value $v \in]u', u[$ cannot be part of any support, hence should be removed from $\mathcal{D}(x_i)$. Moreover, the new upper bound u' has a support on the constraint because BC(D) is maintained. By repeating the process until reaching the lower bound of x_i, we are guaranteed that all values without a support in $\mathcal{D}(x_i)$ are removed. Therefore AC is maintained by using this procedure for every variable x_i. The overall complexity is $O(n \times L)$ since for each variable it takes $O(L)$ to enforce BC(D) from $\max(x_i)$ to $\min(x_i)$. Note that the algorithm is not incremental and the cost of $O(n \times L)$ is for each call to the propagator.

5 On the Complexity of the Hospital/Resident Problem with Forced and Forbidden Pairs

The variant of the Hospital/Resident problem with forced and forbidden pairs (*HRFF*) seeks to find a stable matching that includes or excludes, respectively, a number of pairs $\langle r, h \rangle$ (r denotes a resident and h denotes a hospital). To the best of our knowledge no polynomial algorithm exists in the literature to solve this problem; this observation is also true even if there is no forced pairs [7]. We show that the problem is indeed polynomial and can be solved in $O(L)$ time.

Recall that Theorem 4 states that once Bound(D) consistency is established, then assigning every variable to its maximum is a solution and assigning every variable to its minimum is a solution. Therefore, it is sufficient to enforce Bound(D) consistency on the problem and then take the minimum value in the domain of each variable as the assignment of the correspondent resident. Since BC(D) takes $O(L)$ time, then the complexity of solving HRFF is $O(L)$.

Consider now the particular case of stable marriage with forced and forbidden pairs. There exist a number of polynomial algorithms for solving this problem. The best algorithm for solving this problem runs in $O(n^4)$ time ([2] and Section 2.10.1 in [8]) where n is number of men/women. Now since this problem is a particular case of HRFF, then the complexity for solving it using the above

Table 1. Summary of the Experimental Results

Set	BC(D)-min			AC-min			BC(D)-max		
	Time	Nodes	Opt	Time	Nodes	Opt	Time	Nodes	Opt
2k	2	16.56	**100**	19	13.33	**100**	8	256.38	**100**
4k	5	18.14	**100**	151	14.69	**100**	37	394.86	**100**
6k	9	18.08	**100**	393	14.89	93	86	648.82	**100**
8k	19	18.16	**100**	332	15.80	79	131	491.50	**100**

	AC-max			BC(D)-rand			AC-rand		
	Time	Nodes	Opt	Time	Nodes	Opt	Time	Nodes	Opt
2k	28	204.53	**100**	4	67.36	**100**	12	81.91	**100**
4k	202	320.33	**100**	12	81.91	**100**	160	65.61	**100**
6k	564	535.65	85	25	120.58	**100**	502	99.20	92
8k	530	432.87	69	42	98.88	**100**	475	88.11	77

approach is also $O(L)$. Recall that in the case of stable marriage L is bounded by n^2, therefore the worst case complexity is $O(n^2)$, hence it is optimal.

6 Experimental Results

We consider a variant of the HR problem where some couples can express their desire to be matched together, assuming that they have the same preference lists. The problem is to find a stable matching maximizing the number of such couples who are matched together.

We generated a set of random instances as follows. Each instance is described by a tuple $\langle r, h, c\rangle$ where: $r \in \{2000, 4000, 6000, 8000\}$ is the number of residents; $h \in \{100, 200, 300, 400, 500\}$ is the number of hospitals; and $c \in \{100 + 50 * k \mid k \in [0, 8]\}$ is the number of couples. We implemented the AC and BC(D) propagators in Mistral-2.0 [6]. We use Intel Xeon E5-2640 processors for the experiments running on Linux. The variable ordering is fixed to be lexicographical for all experiments. As for the value ordering, we use three different branching strategies: minimum value (static); maximum value (static); and random min/max. We also use a geometric restart. Note that for random min/max, we use five different seeds since the heuristic is randomized. The time limit is fixed to 20 minutes for each instance and configuration.

The results are shown in Table 1 and Figure 1. The table is split into two parts. In each part, each column depicts one configuration *a*-*b* where *a* is the propagator (BC(D) or AC) and *b* is the value branching strategy. Each row represents the results for one set of instances by their size (i.e. number of residents). We report for each configuration the runtime (Time) in second, the number of nodes (Nodes), and the percentage of instances where optimality where proven (Opt). All the statistics are given as averages across all successful runs (i.e. when optimality is found). Bold faced values show the best results in terms of percent-

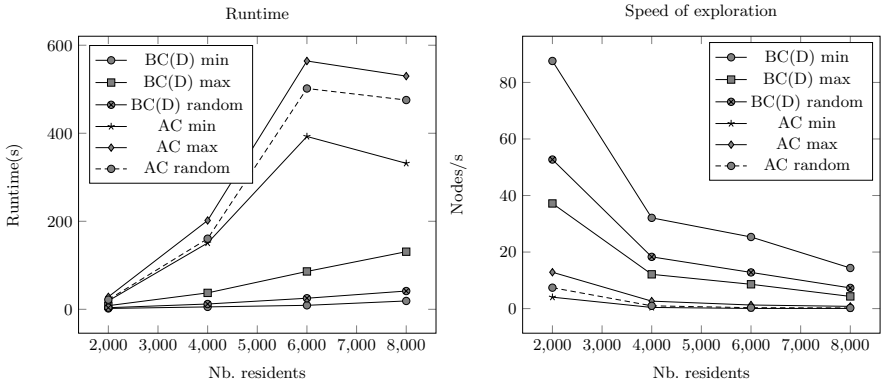


Fig. 1. Runtime and Speed of exploration

age of optimality. Figure 1 shows two plots corresponding to the runtime and the speed of exploration in terms of nodes explored per second.

There are a number of observations based on Table 1 and Figure 1. First, clearly the models enforcing BC(D) outperform the AC models in terms of optimality. For instance, in the 8k set, with min value branching, BC(D) finds the optimal solution for all instances whereas the AC models finds optimality only for 79% of the instances. Second, we observe that enforcing AC considerably slows down the exploration of the search tree. For instance, with set 2k and min value, BC(D) is 21 times faster at exploring the search space (see Figure 1). This behaviour negatively affects the total runtime for AC. For instance, it takes AC 150s to find optimal solutions for the 4k set with min value, whereas the BC(D) models needs about 5s. Last, it should be noted that the search tree is slightly different between the two models. AC does prune some additional nodes as compared to BC(D), but the impact of the pruning does not pay off as it consumes an enormous amount of time. This behaviour is mainly due to the non-incrementality of the AC algorithm.

7 Conclusion

We addressed the filtering aspect of the 2-SidedStability constraint. We first showed that the filtering level of all previous approaches is BC(D). Then, we proposed an optimal BC(D) algorithm, as well as an AC algorithm for this constraint. Our experiments showed that, in practice, BC(D) completely outperforms AC on a variant of the HR problem involving couples. While the aim of this paper was to revisit the current filtering approaches for two-sided stability constraints, we found new theoretical results related to the complexity of the variant of HR with forced and forbidden pairs. We showed, for the first time, that this problem is polynomial and we improved the best known existing complexity for the the particular case of stable marriage with forced and forbidden pairs by a factor of n^2.

References

1. Christian Bessiere. Constraint propagation. In Peter van Beek Francesca Rossi and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 29 – 83. Elsevier, 2006.
2. Vânia M. F. Dias, Guilherme Dias da Fonseca, Celina M. Herrera de Figueiredo, and Jayme Luiz Szwarcfiter. The stable marriage problem with restricted pairs. *Theor. Comput. Sci.*, 306(1-3):391–405, 2003.
3. David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *American mathematical monthly*, pages 9–15, 1962.
4. Ian P. Gent, Robert W. Irving, David Manlove, Patrick Prosser, and Barbara M. Smith. A constraint programming approach to the stable marriage problem. In *Proceedings of CP*, pages 225–239, 2001.
5. Dan Gusfield and Robert W. Irving. *The Stable marriage problem - structure and algorithms*. Foundations of computing series. MIT Press, 1989.
6. Emmanuel Hebrard. Mistral, a Constraint Satisfaction Library. In *Proceedings of the CP-08 Third International CSP Solvers Competition*, pages 31–40, 2008.
7. David Manlove, Gregg O'Malley, Patrick Prosser, and Chris Unsworth. A constraint programming approach to the hospitals / residents problem. In *Proceedings of CPAIOR*, pages 155–170, 2007.
8. David F. Manlove. *Algorithmics of Matching Under Preferences*, volume 2 of *Series on Theoretical Computer Science*. WorldScientific, 2013.
9. Chris Unsworth and Patrick Prosser. A specialised binary constraint for the stable marriage problem. In *Abstraction, Reformulation and Approximation, 6th International Symposium, SARA 2005, Airth Castle, Scotland, UK, July 26-29, 2005, Proceedings*, pages 218–233, 2005.
10. Chris Unsworth and Patrick Prosser. An n-ary constraint for the stable marriage problem. *CoRR*, abs/1308.0183, 2013.