Distributed Stable Marriage Problem *

Ismel Brito and Pedro Meseguer

Institut d'Investigació en Intel.ligència Artificial
Consejo Superior de Investigaciones Científicas
Campus UAB, 08193 Bellaterra, Spain.
{ismel|pedro}@iiia.csic.es

Abstract. The Stable Marriage Problem is a combinatorial problem which can be solved by a centralized algorithm in polynomial time. This requires to make public lists of preferences which agents would like to keep private. With this aim, we define the distributed version of this problem, and we provide a constraint-based approach that solves it keeping privacy. We give empirical results on the proposed approach.

1 Introduction

The Stable Marriage Problem is a classical combinatorial problem, also of interest for Economics and Operations Research. It consists of finding a stable matching between n men and n women, each having his/her own preferences on every member of the other sex. A matching is not stable if there exist a man m and a woman w not matched with each other, such that each of them strictly prefers the other to his/her partner in the matching. Any instance of this problem has a solution, and it can be computed by the centralized Gale-Shapley algorithm in polynomial time.

This problem, by its own nature, appear to be naturally distributed. Each person may desire to act independently. For obvious reasons, each person would like to keep private his/her own preferences. However, in the classical case each person has to follow a rigid role, making public his/her preferences to achieve a global solution. This problem is very suitable to be treated by distributed techniques, trying to provide more autonomy to each person, and to keep preferences private. This paper is a contribution to this aim.

The structure of the paper is as follows. We summarize basic concepts of Stable Marriage and Gale-Shapley algorithm, together with a constraint formulation for this problem from [4] (Section 2). Then, we define the Distributed Stable Marriage problem and provide means to solve it trying to enforce privacy. Thus, we present a distributed version of the Gale-Shapley algorithm, and a distributed constraint formulation under the TCK and PKC models (Section 3). We show how the problem can be solved using the Distributed Forward Checking algorithm with the PKC model, keeping private values and constraints. Experimental results show that, when applicable, the distributed Gale-Shapley

* Supported by the Spanish REPLI project TIC-2002-04470-C03-03.

algorithm is more efficient than the distributed constraint formulation (Section 4), something also observed in the centralized case. However, the distributed constraint formulation is generic and can solve that problem.

2 The Stable Marriage Problem

The Stable Marriage Problem (SM) was first studied by Gale and Shapley [2]. A SM instance consists of two finite equal-sized sets of players, called men and women. Each man m_i ($1 \leq i \leq n$, n is the number of men) ranks women in strict order forming his preference list. Similarly, each woman w_j ($1 \leq j \leq n$) ranks men in strict order forming her preference list. An example of SM appears in Figure 1. A matching M is just a complete one-to-one mapping between the two sexes. The goal is to find a *stable matching* M. A matching M is stable if there is no pair (m, w) of man m and a woman w satisfying the following conditions:

C1. m and w are not married in M,
C2. m prefers w to his current partner in M,
C3. w prefers m to her current partner in M.

If this pair (m, w) exists, M is unstable and the pair (m,w) is called a *blocking pair*. For the example of Figure 1, the matching $M = \{(m_1,w_1),(m_2, w_2), (m_3, w_3)\}$ is not stable because the pair (m_1, w_2) blocks M. For that problem, there is only one stable matching: $M1 = \{(m_1,w_2), (m_2,w_1), (m_3,w_3)\}$. Gale and Shapley showed that each SM instance admits at least one stable matching [2].

A relaxed version of SM occurs when some persons may declare one or more members of the opposite sex to be unacceptable, so they do not appear in the corresponding preference lists. This relaxed version is called the Stable Marriage Problem with Incomplete Lists (SMI). In SMI instances, the goal is also to find a stable matching. Like in the case of SM, SMI admits at least one stable matching. However, some specific features of stable matchings for SMI have to be remarked. First, condition C1 of the definition of *blocking pair* given above have to be changed as follows:

C1'. m and w are not married in M, and m belongs to w's preference list, and w belongs to the m's preference list.

Second, it can not be assured that all the persons can find a partner. So a stable matching of a SMI instance needs not to be complete. However, all the stable matchings involve the same men and women [3].

$m_1 : w_2\ w_3\ w_1$	$w_1 : m_1\ m_2\ m_3$
$m_2 : w_1\ w_2\ w_3$	$w_2 : m_1\ m_3\ m_2$
$m_3 : w_2\ w_1\ w_3$	$w_3 : m_2\ m_1\ m_3$

Fig. 1. A SM instance with three men and three women. Preference lists are in decreasing order, the most-preferred partner is on the left.

```
1.   assign each person to be free;
2.   while some man $m$ is free and $m$ has a nonempty list loop
3.       $w$ := first woman on $m$'s list; {$m$ proposes to $w$}
4.       if $m$ is not on $w$'s preference list then
5.           delete $w$ from $m$'s preference list;
6.           goto line 3
7.       end if
8.       if some man $p$ is engaged to $w$ then
9.           assign $p$ to be free;
10.      end if
11.      assign $m$ and $w$ to be engaged to each other;
12.      for each each successor $p$ of $m$ on $w$'s list loop
13.          delete $p$ from $w$'s list;
14.          delete $w$ from $p$'s list;
15.      end loop;
16.  end loop;
```

Fig. 2. The man-oriented Gale-Shapley algorithm for *SM* and *SMI*.

2.1 The Gale-Shapley algorithm

Gale and Shapley showed that at least one stable matching exists for every *SM* (or *SMI*) instance. They obtained the Gale-Shapley algorithm ([2]), with $O(n^2)$ temporal complexity. The Extended Gale-Shapley algorithm (*EGS*) is an extended version of it, that avoids some extra steps by deleting from the preference lists certain pairs that cannot belong to a stable matching [6]. A man-oriented version of *EGS* [1] appears in Figure 2.

EGS involves a sequence of proposals from men to women. It starts by setting all persons free (line 1). *EGS* iterates until all the men are engaged or, for *SMI* instances, there are some free men because they have an empty preference list (line 2). A man always proposes marriage to his most-preferred woman (line 3). When a woman w receives a proposal from a man m, she accepts it if m is on her preference list. Otherwise, m deletes w from his preference list (line 5) and then a new proposal is started (line 6). Whether m is on w's preference list and w is already engaged to p, she discards the previous proposal with p and p is set free (line 8-9). Afterwards, m and w are engaged each other (line 11). Woman w deletes from her preference list each man p that is less preferred than m (line 13). Conversely, man p deletes w from his preference list (line 14). Finally, if there is a free man with non-empty preference list a new proposal is started. Otherwise, men are engaged or have empty preference lists and the algorithm terminates.

[1] For privacy requirements, that will be discussed later in this work, we prefer not assuming for *SMI* instances, like in [6], that if man m is not acceptable for a woman w, woman w is not acceptable for man m. For avoiding this assumption, we have added Lines 4-7 to the original *EGS*.

$m_1 : w_2$	$w_1 : m_2$
$m_2 : w_1$	$w_2 : m_1$
$m_3 : w_3$	$w_3 : m_3$

Fig. 3. GS-Lists for the *SM* of Figure 1.

During *EGS* execution, some people are deleted from preference lists. The reduced preference lists that result of applying man-oriented Gale-Shapley algorithm are called *man-oriented Gale-Shapley lists* or *MGS-lists*. On termination, each man is engaged to the first woman in his (reduced) list, and each woman to the last man in hers. These engaged pairs constitute a stable matching, and it is called *man-optimal* (or *woman-pessimal*) stable matching since there is not other stable matching where a man can achieve a better partner (according to his ranking). Similarly, exchanging the role of men and women in *EGS* (which means that women propose), we obtain the *woman-oriented Gale-Shapley lists* or *WGS-lists*. On termination, each woman is engaged to the first man in her (reduced) list, and each man to the last woman in his. These engaged pairs constitute a stable matching, and it is called *woman-optimal* (or *man-pessimal*) stable matching.

The intersection of *MGS-lists* and *WGS-lists* is known as the Gale-Shapley lists (*GS-lists*). These lists have important properties (see Theorem 1.2.5 in [6]):

- all the stable matchings are contained in the *GS-lists*,
- in the *man-optimal* (*woman-optimal*), each man is partnered by the first (last) woman on his *GS-list*, and each woman by the last (first) man on hers.

Figure 3 shows the *GS-lists* for the example given in Figure 1. For this instance, the reduced lists of all persons have only one possible partner which means that only one solution exits. In that case, the *man-optimal* matching and *woman-optimal* matching are the same.

2.2 Constraint Formulation

A constraint satisfaction problem (*CSP*) is defined by a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of n variables, $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ is the set of their respective finite domains, and $\mathcal{C}$ is a set of constraints specifying the acceptable value combinations for variables. A solution to a *CSP* is a complete assignment that satisfies all constraints in $\mathcal{C}$. When constraints involve two variables they are called binary constraints.

The *SM* problem can be modeled as a binary *CSP*. In [4] authors propose a constraint encoding for *SM* problems, that we summarize next. Each person is represented by a variable: variables x_1, x_2, ..., x_n represent the men (m_1, m_2, ..., m_n) and variables y_1, y_2, ..., y_n represent the women (w_1, w_2 ..., w_n). $PL(q)$ is the set of people that belong to q's preference list. Domains are as follows:

$$D(x_i) = \{j : w_j \in PL(m_i)\} \quad \forall i,\ 1 \leq i \leq n$$
$$D(y_j) = \{i : m_i \in PL(w_j)\} \quad \forall j,\ 1 \leq j \leq n$$

In the CSP, when variable x_i takes value j, it means that man m_i marries woman w_j. Let be $d_i^m = |D(x_i)|$ and $d_j^w = |D(y_j)|$. Constraints are defined between men and women. Given any pair i and j ($1 \leq i, j \leq n$), the stable marriage constraint x_i/y_j involving x_i and y_j is represented by a $d_i^m \times d_j^w$ conflict matrix C_{ij}. For any pair k, l ($k \in D(x_i)$ and $l \in D(y_j)$), the element $C_{ij}[k, l]$ have one of the following four values:

- $C_{ij}[k, l] =$ **A**llows, when $k = j$ and $l = i$. It allows $x_i = j$ ($y_j = i$). At most one element in C_{ij} is A.
- $C_{ij}[k, l] =$ **I**llegal. This constraint assures the monogamy of the matching, only one man can be married with a woman and vice versa. Entry $C_{ij}[k, l]$ is set to I when either $k = j$ and $l \neq i$ or $k \neq j$ and $l = i$.
- $C_{ij}[k, l] =$ **B**locking pair, when m_i prefers w_j to w_k and w_j prefers m_i to m_l. Since $x_i = j$ blocks pairs $x_i = k$ and $y_j = l$.
- $C_{ij}[k, l] =$ **S**upport, for all the other entries that are not A, I or B.

Figure 4 shows the constraint matrix for man m_3 and woman w_1 of the example given in Figure 1. In the constraint matrix, the domains of x_3 and y_1 are listed in decreasing ordering of the preferences. From that example, we can see that assignment $x_3 = w_1$ does not block any of other pairs which involve variable x_3 or variable y_1.

	m_1	m_2	m_3
w_2	S	S	I
w_1	I	I	A
w_3	S	S	I

	m_1	m_2	m_3
w_2	1	1	0
w_1	0	0	1
w_3	1	1	0

Fig. 4. C_{31} for example of Figure 1. Left: in terms of A,I,B,S. Right: in terms of 0/1.

To encode SMI instances, it is needed to add a dummy man m_{n+1} and a dummy woman w_{n+1} to the problem. Man and woman variables remain the same but their domains are enlarged with the value $n + 1$, that is always the least preferred one. Whether a person p is not an accepted partner for a person q, of opposite sex, all entries in column or row assigning p to q on C_{pq} are I. The rest of the constraint table is filled with S.

Constraint tables in terms of A, I, B, S are transformed in terms of 1/0 (permitted/forbidden) pairs, using the natural conversion A, S $\rightarrow$ 1, I, B $\rightarrow$ 0.

In [4], it is shown that encoding a SMI instance I as a CSP instance J produces all stable matchings of I as solutions of J. It is easy to show that J has no more solutions. Special emphasis is put on the fact that achieving arc consistency on J produces a reduced domains which are exactly the $GS - lists$ obtained by the EGS algorithm.

3 The Distributed Stable Marriage Problem

The Distributed Stable Marriage problem ($DisSM$) is defined as in the classical (centralized) case by n men $\{m_1, \ldots, m_n\}$ and n women $\{w_1, \ldots, w_n\}$, each having a preference list where all members of the opposite sex are ranked, plus a set of r agents $\{a_1, \ldots, a_r\}$. The n men and n women are distributed among the agents, such that each agent owns some persons and every person is owned by a single agent. An agent can access and modify all the information of the owned persons, but it cannot access the information of persons owned by other agents. To simplify description, we will assume that each agent owns exactly one person (so there are $2n$ agents). As in the classical case, a solution is a stable matching (a matching between the men and women such that no blocking pair exists). A complete stable matching always exists.

Analogously to the classical case, we define the Distributed Stable Marriage with Incomplete lists problem ($DisSMI$) as a generalization of $DisSM$ that occurs when preference lists do not contain all persons of the opposite sex (some options are considered unacceptable). A solution is a stable matching, and it always exists, although it is not guaranteed to be a complete one (some men/women may remain unmatched).

This problem, by its own nature, appears to be naturally distributed. First, each person may desire to act as an independent agent. Second, for obvious reasons each person would like to keep private his/her preference list ranking the opposite sex options. However, in the classical case each person has to follow a rigid role, making public his/her preferences to achieve a global solution. Therefore, this problem is very suitable to be treated by distributed techniques, trying to provide more autonomy to each person, and to keep private the information contained in the preference lists.

3.1 The Distributed Gale-Shapley Algorithm

The EGS algorithm that solves the classical SMI can be easily adapted to deal with the distributed case. We call this new version the Distributed Extended Gale/Shapley ($DisEGS$) algorithm. As in the classical case, the $DisEGS$ algorithm has two phases, the man-oriented and the woman-oriented, which are executed one after the other. Each phase produces reduced preference lists for each person. The intersection of these lists produces a $GS - list$ per person. As in the classical case, the matching obtained after executing the man-oriented phase is a stable matching (*man-optimal*).

The man-oriented version of the $DisEGS$ algorithm appears in Figure 5 (the woman-oriented is analogous, switching the roles man/woman). It is composed of two procedures, `Man` and `Woman`, which are executed on each man and woman, respectively. Execution is asynchronous. The following messages are exchanged (where m is the man that executes `Man` and w the woman that executes `Woman`),

- propose: m sends this message to w to propose engagement;

```
procedure Man()
$m \leftarrow free$;
$end \leftarrow$ false;
while $\neg end$ do
  if $m = free$ and $list(m) \neq \emptyset$ then
    $w \leftarrow first(list(m))$;
    sendMsg(propose,$m$,$w$);
    $m \leftarrow w$;
  $msg \leftarrow$ getMsg();
  switch $msg.type$
    accept  : do nothing;
    delete  : $list(m) \leftarrow list(m) - msg.sender$;
              if $msg.sender = w$ then $m \leftarrow free$;
    stop    : $end \leftarrow$ true;

procedure Woman()
$w \leftarrow free$;
$end \leftarrow$ false;
while $\neg end$ do
  $msg \leftarrow$ getMsg();
  switch $msg.type$
    propose: $m \leftarrow msg.sender$;
             if $m \notin list(w)$ then
               sendMsg(delete,$w$,$m$);
             else
               sendMsg(accept,$w$,$m$);
               $w \leftarrow m$;
               for each $p$ after $m$ in $list(w)$ do
                 sendMsg(delete,$w$,$p$);
                 $list(w) \leftarrow list(w) - p$;
    stop    : $end \leftarrow$ true;
```

Fig. 5. The man-oriented version of the *DisEGS* algorithm.

– accept: w sends this message to m after receiving a propose message to notify acceptance;
– delete: w sends this message to m to notify that w is not available for m; this occurs either (i) proposing m an engagement to w but w has a better partner or (ii) w accepted an engagement with other man more preferred than m;
– stop: this is an special message to notify that execution must end; it is sent by an special agent after detecting quiescence.

Procedure Man, after initialization, performs the following loop. If m is free and his list is not empty, he proposes to be engaged to w, the first woman in his list. Then, m waits for a message. If the message is accept and it comes from w, then m confirms the engagement (nothing is done in the algorithm). If the message is delete, then m deletes the sender from his list, and if the sender is w then m becomes free. The loop ends when receiving a stop message.

Procedure `Woman` is executed on woman w. After initialization, there is a message receiving loop. In the received message comes from a man m proposing engagement, w rejects the proposition if m is not in her list. Otherwise, w accepts. Then, any man p that appears after m in w list is asked to delete w from his list, while w removes p from hers. This includes a previous engagement m', that will be the last in her list. The loop ends when receiving a stop message.

Algorithm *DisEGS* is a distributed version of *EGS*, where each person can access to his/her own information only. For this reason there are two different procedures, one for men and one for women. In addition, actions performed by *EGS* on persons different from the current one are replaced by message sending. Thus, when m assigns woman w is replaced by `sendMsg`(propose,m,w); when w deletes herself from the list of p is replaced by `sendMsg`(delete,w,p). Since procedures exchange messages, operations of message reception are included accordingly.

DisEGS algorithm guarantees privacy in preferences and in the final assignment: each person knows the assigned person, and no person knows more than that. In this sense, it is a kind of ideal algorithm because it assures privacy in values and constraints.

3.2 Distributed Constraint Formulation

In [1] we presented an approach to privacy that differentiates between values and constraints. Briefly, privacy on values implies that agents are not aware of other agent values during the solving process and in the final solution. This was achieved using the Distributed Forward Checking algorithm (*DisFC*), an *ABT*-based algorithm that, after the assignment of an agent variable, instead of sending to lower priority agents the value just assigned, it sends the domain subset that is compatible with the assigned value. In addition, it replaces actual values by sequence numbers in backtracking messages. In this way, the assignment of an agent is kept private at any time.

Regarding privacy on constraints, two models were considered. The Totally Known Constraints (*TKC*) model assumes that when two agents i, j share a constraint C_{ij}, both know the constraint scope and one of them knows completely the relational part of the constraint (the agent in charge of constraint evaluation). The Partially Known Constraints (*PKC*) model assumes that when two agents i, j share a constraint C_{ij}, none of them knows completely the constraint. On the contrary, each agent knows the part of the constraint that it is able to build, based on its own information. We say that agent i knows $C_{i(j)}$, and j knows $C_{j(i)}$. This implies that the identification of the other agent is neutral. In the following, we apply these two models to the *DisSM* problem using the constraint formulation of Section 2.2 from [4].

Totally Known Constraints Solving a *DisSMI* problem is direct under the *TKC* model. For each pair x_i, y_j, representing a man and a woman, there is a

$$\begin{array}{cccccccc} & & & & i & & & \\ & 1 & \dots & 1 & 0 & 1 & \dots & 1 \\ & & \dots & & & & & \\ & 1 & \dots & 1 & 0 & 1 & \dots & 1 \\ j & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ & 1 & \dots & 1 & 0 & 0 & \dots & 0 \\ & & \dots & & & & & \\ & 1 & \dots & 1 & 0 & 0 & \dots & 0 \end{array} \qquad \begin{array}{cccccccc} & & & & i & & & \\ & 1/0 & \dots & 1/0 & 0 & 1/0 & \dots & 1/0 \\ & & \dots & & & & & \\ & 1/0 & \dots & 1/0 & 0 & 1/0 & \dots & 1/0 \\ j & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ & 1/0 & \dots & 1/0 & 0 & 1/0 & \dots & 1/0 \\ & & \dots & & & & & \\ & 1/0 & \dots & 1/0 & 0 & 1/0 & \dots & 1/0 \end{array}$$

Fig. 6. Constraint C_{ij}. Left: rows and columns ordered by decreasing preferences of x_i and y_j, respectively. Right: rows and columns ordered lexicographically.

constraint C_{ij} that appears in Figure 6. We assume, without loss of generality, that agents owning men have higher priority than agents owning women. Using $DisFC$, constraint C_{ij} has to be known by the agent of variable x_i. Conversely, using ABT, constraint C_{ij} has to be known by the agent owning y_j. If an agent knows C_{ij}, it can deduce the preferences of the other agent.

Using ABT, there is no privacy of values. Using $DisFC$, there is privacy of values, since values are never made public to other agents. This model does not allow privacy of constraints.

Partially Known Constraints A $DisSMI$ instance can be formulated in the PKC model as follows. The partially known constraint $C_{i(j)}$ is built from x_i, knowing its preference list but ignoring the preference list of y_j. Analogously, $C_{(i)j}$ is built knowing the preference list of y_j but ignoring the preference list of x_i. Assuming lexicographical ordering in rows and columns, they are of the form,

$$C_{i(j)} = \begin{array}{cccccccc} & & & & i & & & \\ & 1/? & \dots & 1/? & 0 & 1/? & \dots & 1/? \\ & & \dots & & & & & \\ & 1/? & \dots & 1/? & 0 & 1/? & \dots & 1/? \\ j & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ & 1/? & \dots & 1/? & 0 & 1/? & \dots & 1/? \\ & & \dots & & & & & \\ & 1/? & \dots & 1/? & 0 & 1/? & \dots & 1/? \end{array} \quad C_{(i)j} = \begin{array}{cccccccc} & & & & i & & & \\ & 1/? & \dots & 1/? & 0 & 1/? & \dots & 1/? \\ & & \dots & & & & & \\ & 1/? & \dots & 1/? & 0 & 1/? & \dots & 1/? \\ j & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ & 1/? & \dots & 1/? & 0 & 1/? & \dots & 1/? \\ & & \dots & & & & & \\ & 1/? & \dots & 1/? & 0 & 1/? & \dots & 1/? \end{array}$$

where 1/? means that the value can be either 1 (permitted) or ? (undecided). Undecided values appear in $C_{i(j)}$ (conversely $C_{(i)j}$) because x_i (y_j) does not know the preference list of y_j (x_i). As example, the partially known constraints corresponding to the constraint of Figure 4 appear in Figure 7.

One interesting property of these constraints is that in $C_{i(j)}$ (conversely $C_{(i)j}$) all columns (rows) are equal, except the column (row) corresponding to x_i (y_j).

Proposition 1. *In $C_{i(j)}$ (conversely $C_{(i)j}$) all columns (rows) are equal, except the column (row) corresponding to x_i (y_j).*

$$C_{3(1)} = \begin{array}{cccc} & m_1 & m_2 & m_3 \\ w_1 & 0 & 0 & 1 \\ w_2 & 1 & 1 & 0 \\ w_3 & ? & ? & 0 \end{array} \quad C_{(3)1} = \begin{array}{cccc} & m_1 & m_2 & m_3 \\ w_1 & 0 & 0 & 1 \\ w_2 & 1 & 1 & 0 \\ w_3 & 1 & 1 & 0 \end{array}$$

Fig. 7. Partially known constraints of constraint of Figure 4.

Proof. We have to prove that $C_{i(j)}[k,l] = C_{i(j)}[k,l'], l \neq i, l' \neq i, l \neq l'$. Effectively, if x_i prefers woman k to woman j, both values $C_{i(j)}[k,l]$ and $C_{i(j)}[k,l']$ are 1, corresponding to S (supported, see Section 2.2). If x_i prefers woman j to woman k, both values $C_{i(j)}[k,l]$ and $C_{i(j)}[k,l']$ are ? (undecided). Their exact value could be 1 or 0, depending on the preferences of y_j, information which is not available when constructing $C_{i(j)}$. Therefore, both are undecided in $C_{i(j)}$. An analogous argument holds for $C_{(i)j}$ rows. □

It is interesting to observe the relation between $C_{i(j)}, C_{(i)j}$ and C_{ij}. It is easy to check that $C_{ij} = C_{i(j)} \diamond C_{(i)j}$, where $\diamond$ operates component to component with the following rules,

$$\begin{array}{lll} 1 \diamond 1 = 1 & 1 \diamond 0 = error & 0 \diamond 0 = 0 \\ ? \diamond 1 = 1 & ? \diamond 0 = 0 & ? \diamond ? = 0 \end{array}$$

Rules including ? are quite intuitive (if a position in the constraint is decided (permitted/forbidden) in one constraint and undecided in the other, the result is the decided value). The last rule $?\diamond? = 0$ is proved next.

Proposition 2. *If entry $[k,l]$ is undecided in both partially known constraints for variable x_i and variable y_j ($C_{i(j)}[k,l] =?$ and $C_{(i)j}[k,l] =?$), then entry $[k,l]$ is 0 in the complete constraint table ($C_{ij}[k,l] = 0$).*

Proof. From the construction of partially known constraints, all undecided entries in $C_{i(j)}$ are related to values which are less preferred than j. If $C_{i(j)}[k,l] =$?, we can infer that x_i prefers j to k. Conversely, if $C_{(i)j}[k,l] =?$, we infer that y_j prefer i to l. Therefore, since x_i prefers j to k and y_j prefers i to l, the pair (i,j) is blocking pair to the pair (k,l) so $C_{ij}[k,l] = 0$. □

With these properties, we can specialize the *DisFC* algorithm to solve *DisSMI* instances, using phase I only. This specialized algorithm works as follows. Each agent instantiates its variable and sends to lower priority agents the domains compatible with its assignment. For *DisSMI*, after assignment each man agent sends to each woman agent the domain that she can take. A woman agent receives messages from every man agent, and assigns a value permitted by these n received domains. If no value is available, the woman agent performs backtracking. The process iterates until finding a solution. Since *DisFC* is a complete algorithm, and the constraint encoding of *SMI* is correct (a *SMI* solution corresponds one-to-one with solutions of the constraint encoding), a solution will be found.

In the previous argument, something must be scrutinized in more detail. After assignment, what kind of compatible domain can a man agent send? If agent i assigns value k to x_i, it sends to j the row of $C_{i(j)}$ corresponding to value k. This row may contain 1's (permitted values for y_j), 0's (forbidden values for y_j), and ? (undecided values for y_j). If the compatible domain has 1 or 0 values only, there is no problem and the y_j domain can be easily computed. But what happens when the domain contains ? (undecided) values? In this case, agent j can disambiguate the domain as follows. When agent j receives a compatible domain with ? (undecided) values, it performs the $\diamond$ operation with a row of $C_{(i)j}$ different from i. Since all rows in $C_{(i)j}$ are equal, except row corresponding to value i (see Proposition 1), all will give the same result. Performing the $\diamond$ operation j will compute the corresponding row in the complete constraint C_{ij}, although j does not know to which value this row corresponds (in other words, j does not know the value assigned to x_i). After the $\diamond$ operation the resulting received domain will contain no ? (undecided) values, and the receiving agent can operate normally with it.

4 Experimental Results

We give empirical results on the *DisEGS* and *DisFC* algorithm with *PKC* model for *DisSMI* random instances. In our experiments, we use four different classes of instances. Each class is defined by a pair $\langle n, p_1 \rangle$, which means the problem has n men and n women on *DisSMI* and the probability of incompleteness of the preference list is p_1. Class $< n, 0.0 >$ groups instances where all the n persons has complete preference list. If $p_1 = 1.0$, preferences lists are empty. We study 4 different classes with $n = 10$ and $p_1 = 0.0$, 0.2, 0.5 and 0.8. For each class, 100 instances were generated following the random instance generation presented in [5] considering p_2, the probability of ties, equals to 0.0.

Algorithm is modeled using $2n$ agents, each one represents a man or a woman. In *DisEGS*, each agent only knows its preference list. In *DisFC*, each agent only knows its partial constraint tables. In both, agents exchange different kind of messages to find a stable matching. When *DisEGS* finishes the stable matching found is the *men-optimal* one. *DisFC*, like others asynchronous backtracking algorithms, requires a total ordering among agents. In our experiments, men agents have higher priority than women agents.

Algorithmic performance is evaluated by communication and computation effort. Communication effort is measured by the total number of exchanged messages (*msg*). Since both algorithms are asynchronous, computation cost is measured by the number of concurrent constraint checks (*ccc*) (for *DisEGS*, an agent performs a constraint check when it checks if one person is more preferred that other), as defined in [7], following Lamport's logic clocks [8].

Table 1 details the experimental results of *DisEGS* on *DisSMI*. Besides *msg* and *ccc*, we provide the total number of messages for each kind of message and the total number of checks. Regarding the communication effort, except for instances with $p_1 = 0.8$, the larger number of exchanged messages are for

p_1	propose	accept	delete	msg	checks	ccc
0.0	27.8	22.8	65.6	98.4	133.6	52.7
0.2	28.1	22.8	53.0	86.4	107.9	43.0
0.5	26.3	21.9	32.7	63.4	67.3	27.7
0.8	20.9	18.5	12.1	35.4	28.0	10.6

Table 1. Results of $DisEGS$ for solving $DisSMI$ instances with 10 men, 10 women and with $p_1 = 0.0$, 0.2, 0.50 and 0.8.

p_1	info	back	link	msg	checks	ccc
0.0	39,686	5,987	72	45,745	4,833,478	3,153,363
0.2	31,636	5,272	62	36,970	3,941,676	2,580,822
0.5	4,324	840	48	5,212	355,436	223,469
0.8	222	47	27	296	10,022	5,651

Table 2. Results of $DisFC$ for solving $DisSMI$ instances with 10 men, 10 women and with $p_1 = 0.0$, 0.2, 0.50 and 0.8.

deleting persons from preference lists. When $p_1 = 0.2$, women agents receive more proposals than women agents for $p_1 = 0.0$. In general, the number of proposals, proposal acceptances and the total number of messages tend to decrease when p_1 increases. Considering the computation effort of $DisEGS$, the larger values of p_1 (shorter lists), the easier instances. In general, the communication and computation costs of $DisEGS$ are the largest with complete lists.

Table 2 resumes the experimental results of $DisFC$ on $DisSMI$. We observe the same trend as in $DisEGS$ results: instances with complete lists are the most difficult to solve (both in terms of *msg* and *ccc*) and they become easier to solve as lists become shorter. According to the reported results, $DisFC$ is much worse than $DisEGS$. This could be expected, since $DisEGS$ as specialized algorithm for this particular problem, takes advantage of the problem features. $DisFC$ is a generic algorithm that is applicable to any CSP. When applied to this problem, a tractable CSP, it gets worse results than the specialized approach. Nevertheless, the distributed constraint formulation is generic and it is potentially applicable to further generalizations of this problem (i.e. the Stable Roommates Problem [6]) which are computationally harder to solve.

5 Conclusions

We presented a distributed formulation for the Stable Marriage problem. This problem appear to be naturally distributed and there is a clear motivation to keep its preference lists private during the solving process. A distributed version of the centralized algorithm can efficiently solve the problem, keeping the required privacy. Using the constraint formulation of [4], we have provided a simple way to solve this problem using the Distributed Forward Checking algorithm with the Partially Known Constraint model, keeping the required privacy. We provide experimental results of this approach. The generic constraint formulation opens new directions for distributed encodings of harder versions of this problem.

References

1. Brito, I. and Meseguer, P. Distributed Forward Checking. *Proc. CP-2003*, 801–806, 2003.
2. Gale D. and Shapley L.S. College admissions and the stability of the marriage. *American Mathematical Monthly.* 69:9–15, 1962.
3. Gale D. and Sotomayor M. Some remarks on the stable matching problem. *Discrete Applied Mathematics.* 11:223–232, 1985.
4. Gent, I. P. and Irving, R. W. and Manlove, D. F. and Prosser, P. and Smith, B. M. A constraint programming approach to the stable marriage problem. *Proc. CP-2001*, 225-239, 2001.
5. Gent,I. and Prosser,P. An Empirical Study of the Stable Marriage Problem with Ties and Incomplete Lists. *Proc. ECAI-2002*, 141–145, 2002.
6. Gusfield D. and Irving R. W. *The Stable Marriage Problem: Structure and Algorithms.* The MIT Press, 1989.
7. Meisels A., Kaplansky E., Razgon I., Zivan R. Comparing Performance of Distributed Constraint Processing Algorithms. *AAMAS-02 Workshop on Distributed Constraint Reasoning*, 86–93, 2002.
8. Lamport L. Time, Clock, and the Ordering of Evens in a Distributed System. *Communications of the ACM*, **21(7)**, 558–565, 1978.