Time Efficient Self-stabilizing Stable Marriage

Joffroy Beauquier, Thibault Bernard, Janna Burman, Shay Kutten, Marie Laveau

Time Efficient Self-stabilizing Stable Marriage

Joffroy Beauquier[1], Thibault Bernard[2], Janna Burman[1], Shay Kutten[3], and Marie Laveau*[1]

[1]LRI, Université Paris-Sud, CNRS, Université Paris-Saclay, Orsay , France
[2]LI-PaRAD, Université de Versailles, Université Paris-Saclay, Versailles, France
[3]Technion - Israel Institute of Technology, Haifa, Israel

Abstract

"Stable marriage" refers to a particular matching with constraints, introduced in the economic context of two-sided markets. The problem has a wide variety of other applications in different domains, such as Cloud computing, Internet content delivery or college admissions. Most of the solutions known up to now are given for the synchronous model, in which executions proceed in rounds, and assume initialization. In this paper, we consider a distributed and asynchronous context, without initialization (*i.e.*, in a self-stabilizing manner - tolerating any transient faults) and with some confidentiality requirements. The single already known self-stabilizing solution [24], based on Ackerman et al.'s algorithm [1], is in $O(n^4)$ moves (activation of a single node). We considerably improve on this previous result by presenting a solution with an complexity of $O(n^2)$ moves (and $O(n^2)$ asynchronous rounds), relying on Gale and Shapley's algorithm [14]. This algorithm runs also in $O(n^2)$ moves, but in a centralized synchronous context. Moreover it is not self-sabilizing and a corruption cannot be repaired locally, as noticed by Knuth [21].

1 Introduction

In this paper, we are interested in a matching problem on complete bipartite graphs. This problem was originally called stable marriage by Gale and Shapley [14]. It was first introduced in economics and can be described informally as follows. There are two sets of equal size. Here, the first set contains n men and the other n women. Women have preferences for men and men have preferences for women, in the form of preference lists. The list of a woman starts from her

*This work was supported in part by grants from Digiteo France.

most preferred man, continues with the second one and so on. The problem consists in matching women and men, in such a way that there is no unmarried pair (woman, man) with both partners preferring each other to their current spouses. Such a matching is said to be without *blocking pairs* (the term comes from Gale and Shapley, but "unstable pair" might have been more appropriate).

In the light of game theory, one would say that a stable marriage realizes a pure Nash equilibrium, given lists of preferences for both women and men [1]. For example, the taxi scheduling problem can be considered as a game and finding the equilibrium can help taxi drivers to make decisions in their work [5]. Furthermore, simulations have shown that solving stable matching for the scheduling increases the benefits of taxi drivers and decreases the waiting time of passengers [22]. The problem has also received a considerable attention in economics, because it abstracts a basic situation with producers and consumers in two-sided matching markets. Solutions are widely used all over the world, to assign students to hospitals, schools or universities [15]. In the domain of distributed computing, stable marriage has a lot of applications. For instance, Cloud Computing needs efficient migration algorithms, that work well with thousands of Virtual Machines (VM) and servers. Many stable marriage algorithms have been proposed to math servers and VMs in the Cloud (*cf.* [19, 29]). Another important and large-scale application of stable marriage is in assigning users to servers in a distributed Internet service [25]. To access web pages, videos, and other services on the Internet, each user has to be matched to one of the numerous servers around the world that offer that service. Users prefer servers that are near to get a faster response time and servers prefer to serve users with a lower cost. Content delivery networks that distribute much of the world's content and services have to solve a large and complex stable matching problem between users and servers. One can also note that stable marriage has applications in models without any hint of selfish agents, such as scheduling network switches [9], and many others. This problem has also been theoretically well studied and the reference book [17] surveys the problem and some of its variants.

The first solution to the stable matching problem was proposed by Gale and Shapley [14]. This algorithm (GSA) is centralized and proceeds in synchronous rounds, alternating proposals (by women) and acceptances (by men). Intuitively speaking, one can say that the algorithm tries to avoid blocking pairs by gradually improving the quality of the matchings (men "better match" dynamics). As noticed by Knuth in [21], GSA requires an initial configuration in which no node is matched, meaning that it is not *self-stabilizing* (see the definition in Sec. 2.3). As self-stabilization is a way of tolerating transient failures and as many applications of stable marriage, especially in computer science, require failure tolerance, it seemed natural to look for a self-stabilizing solution. Such a solution was proposed in [24]. It is distributed, correct in an asynchronous context and guarantees some confidentiality, in the sense that the list of preferences of a node is not communicated to the others. Its complexity is $O(n^4)$ moves, in contrast to that of GSA, $O(n^2)$, although with different assumptions. In addition, to ensure confidentiality of the preferences [6] and avoid high commu-

nication complexity, we follow previous mentioned studies and rule out a trivial solution where nodes exchange their preference lists and then run a known centralized solution at each node. This solution would give a move complexity of $O(n)$ moves, which is unattainable without the exchanging of the preference lists.

Contribution The proposed solution greatly improve the solution of [24] under the same assumptions (distributed, self-stabilizing, asynchronous, confidential), providing an complexity of $O(n^2)$ moves. The considered model is even more general (shared registers instead of state sharing (composite atomicity) in [24]). This result is not incremental. While the solution of [24] in $O(n^4)$ moves is inspired by a two-phase algorithm due to Ackerman et al. [1], the proposed solution relies on GSA. Making GSA self-stabilizing without augmenting its complexity is a challenge, because GSA is inherently not self-stabilizing: starting GSA from some configurations leads to cycles [21]. As we want to keep the "better match" dynamics of GSA because it ensures an good complexity, we are naturally led to detect locally the blocking pairs and to repair them globally, since no local repair is possible. Once a repair is made, the solution has to avoid the formation of new blocking pairs, which would provoke a new activation of the repair mechanism and so on. Our solution acts according to this scheme, which allows to get the complexity of $O(n^2)$ moves. The lower bound $O(n^2)$ moves has been proved in the more powerful centralized setting and then applies to the distributed that is considered here. More technically, the solution follows a scheme proposed in [4] and relies on two modules. First, a detection module checks locally the presence of blocking pairs and the correctness of the configuration. If a problem is detected, the module triggers a reset reinitializing the system. We use the self-stabilizing reset given in [3], propagated on a spanning tree over the given bipartite graph. This first module stabilizes in $O(n^2)$ moves. Then, a second module builds a stable marriage from the reinitialized system, in such a way that no blocking pair is created during the process, nor the detection module triggered again. For this latter module, we develop an algorithm, `Async-GSA`, for building a stable marriage. The algorithms are given with a proof of correctness and a complexity analysis.

Additional related work Most of the publications on stable marriage concern centralized versions and are less relevant here. Studies on distributed stable marriage appeared much later and usually consider a *synchronous* distributed communication model, where nodes progress in a lock-step manner, exchanging information and performing computations *all* together at each step (called *round*). These studies focus on the round complexity of the problem and its variants. Kipnis and Patt-Shamir [20] proved a lower bound of $\Omega(\sqrt{(n/B \log n)})$, where B is the number of bits per message, and provided an algorithm that solves the distributed stable marriage in $O(n^2)$ rounds. Searching for better time complexity and conditions that can provide it, many studies considered specific restrictions on the preference lists such as *weighted stable marriage* [2],

incomplete or bounded lists [13, 27], "almost regular" lists [27] and "similarity" in preference lists [18]. Still for improving time complexity, approximate versions have been considered (*e.g.*, [13, 16, 20, 27]) and reach a polylogarithmic time. Furthermore, when assuming string restrictions on preference lists, approximate stable marriage can be solved even in constant time (*cf.* [13, 27]). Note also several bound results on its communication and step complexity (*cf.* [8, 16, 28]). Apart of the already mentioned [24], there is no other work addressing the asynchronous, distributed and self-stabilizing case.

2 Preliminaries

2.1 Distributed stable marriage problem

A distributed system is based on a set of nodes. Each node v can communicate (directly) with a subset of other nodes, called its neighbors, denoted by $\mathcal{N}(v)$ (not including v). Communication is assumed to be bidirectional. Hence, the topology of the system can be represented as a simple undirected graph $G = (V, E)$, where V is the set of nodes and E the set of edges, *i.e.*, communication links. It is assumed that G is a complete bipartite graph $K_{n,n}$, over two subsets of nodes of equal size. We are interested in the *stable marriage* problem. Following the terminology of [14], where the problem is introduced, we call women the n nodes of the first subset (Women) in the bipartite graph and men the n nodes of the second subset (Men). Each node knows its gender, has a unique identifier and a complete list of n preferences for the nodes of the other set (each woman has a complete list of men and symmetrically). In other words, each women w is given a *priority* for each man m, denoted $priority(w, m)$, and reciprocally. The priorities go from 1 to n and the most preferred person has priority 1.

The goal is to match (marry) the women and the men together such that everyone is matched and there is no unmarried pair (w, m) of a woman and a man, who both prefer each other to their current matches (partners) m' and w', *i.e.*, there is no pair (w, m) such that (w, m') and (w', m) are married, but $priority(w, m) < priority(w, m')$ and $priority(m, w) < priority(m, w')$. When there are no such pairs of people, called *blocking pairs* (BP), the set of marriages is said *stable*.

2.2 Model

For designing solutions to this problem, we use the link-register communication model (*cf.* [11]) in which each process v is associated with a set of atomic registers. A process v can write in its associated registers and can read any register on adjacent communication links. We use the notation $r_{v,u}$ to refer to the register of v on link (v,u). Each register contains many variables named $var_{v,u}$ for variable var in register $r_{v,u}$. It can only be written by v and can be read by both v and u.

The *state* of a node is a vector of the values of its variables (local and shared). A *configuration* of the system is a vector of states of all nodes. A distributed algorithm consists of one program per node. The program of a node v is a finite set of guarded rules of the following form:

Label: (* Comment *)
{Guard}
Actions

The labels are used to identify rules. The guard of a rule in the code of v is a Boolean expression involving variables of v and registers of its neighbors. If the guard of some rule evaluates to true, then the rule is said to be *enabled* at v. By extension, v is said to be enabled or *eligible* if at least one of its rules is enabled. *Actions* represent a sequence of actions on v's variables. A rule can be executed (activated) only if it is enabled. In this case, its execution consists in performing the sequence of actions, using the values of the variables at the time of the guard evaluation. The asynchrony of the system is modeled by an adversary, called *scheduler*. In a configuration, the scheduler selects a non-empty subset of eligible nodes, then atomically evaluates the guard of one enabled rule per node (chosen non-deterministically), then, still atomically, executes the corresponding actions. This is called a *step* (or *transition*) and the activation of each rule in a step is called a *move*. Such a scheduler is called *distributed* in the literature (contrary to a *central* scheduler, choosing at each step only one enabled node, or to the *synchronous* scheduler that chooses all the enabled nodes). When a step is executed in the configuration C, it leads to a configuration C' and we write $C \rightarrow C'$. We say that C' is *reached* from C, denoted by $C \xrightarrow{*} C'$, if $C \rightarrow C_1 \rightarrow C_2 \rightarrow ... \rightarrow C'$. An *execution* is a maximal sequence of configurations C_0, C_1, ... , C_k, ... such that $C_i \rightarrow C_{i+1}$ for all $i \geq 0$. The term "*maximal*" means that the execution is either infinite or ends in a *terminal configuration*, *i.e.*, a configuration in which no node is enabled. Different types of fairness, limiting the possible choices of the scheduler, appear in the literature. We do not make any such limitation (except the obviously necessary one, obligating the scheduler to chose at least one eligible node at each step), that is the schedulers we consider are *unfair*.

Furthermore, we use the notion of asynchronous round introduced in [12], extended with the concept of *neutralization* [10]. A process v is said *neutralized* during a step $\gamma_i \rightarrow \gamma_{i+1}$, if v is enabled in configuration γ_i but not in configuration γ_{i+1}, and is not activated in the step $\gamma_i \rightarrow \gamma_{i+1}$. The rounds are inductively defined as follows. The first round of an execution $e = \gamma_0, \gamma_1, \ldots$ is the minimal prefix $e' = \gamma_0, \ldots, \gamma_j$, such that every process that is enabled in γ_0 either executes a rule or is neutralized during a step of e'. Let e'' be the suffix $\gamma_j, \gamma_{j+1}, \ldots$ of e. The second round of e is the first round of e'', and so on.

A distributed algorithm *solves* the stable marriage problem if each of its executions starting from a predefined initial configuration, under the unfair distributed scheduler, reaches a terminal configuration in which there is a stable marriage.

2.3 Self-stabilization

A distributed algorithm solves the stable marriage problem in a *self-stabilizing* way if it solves it as above, but for any possible initial configuration. The relation between self-stabilization and transient failures is well known. Even if all the variables of all nodes have been corrupted once, (producing an arbitrary configuration possibly considered as initial), the algorithm reaches a terminal configuration in which there is a stable marriage. Hence, in some sense, it tolerates transient failures, since it regains a correct configuration by itself, without any external intervention. Formally, let $\mathcal{A}$ be a distributed algorithm, let $\mathcal{C}$ be the set of its configurations and let $\mathcal{E}$ be the set of its executions, from any configuration in $\mathcal{C}$. Call *graph problem* a predicate $\mathcal{P}$ on configurations.

Definition 1. *$\mathcal{A}$ is self-stabilizing for $\mathcal{P}$ if and only if there exists a non-empty subset $\mathcal{L}$ of configurations of $\mathcal{C}$, such that:*

1. *(Closure) starting from any $C \in \mathcal{L}$, any reachable configuration is in $\mathcal{L}$ (i.e., $\mathcal{L}$ is closed under $\rightarrow$) and every configuration in $\mathcal{L}$ satisfies $\mathcal{P}$,*
2. *(Convergence) any execution in $\mathcal{E}$ (starting from any configuration in $\mathcal{C}$), reaches a configuration in $\mathcal{L}$.*

The time complexity of a self-stabilizing distributed algorithm can be evaluated in terms of moves or steps. The *stabilization time* of a distributed algorithm, counted in moves (respectively in steps), is the maximum number of moves (resp. steps) until a configuration in $\mathcal{L}$ is reached, starting from an arbitrary configuration. The stabilization time in moves gives an upper bound on the stabilization time in steps.

3 Asynchronous (non-self-stabilizing) stable marriage

Based on the Gale and Shapley's algorithm [14], we propose a distributed and asynchronous algorithm that solves the stable marriage problem. This is not a self-stabilizing algorithm yet, but it is transformed to such in the next section. The algorithm works as follows. From an initial configuration $\mathcal{C}_{init}$, each woman proposes to men in the order of her preference list, starting from the first element. Doing so ensures that no blocking pair appears in the final matching (and during the whole execution). Men reply by accepting or refusing each proposal. If a man receives several proposals, he accepts the best one and refuses the others. If he is already married but receives a better proposal, he accepts it and cancels the previous marriage. Following which, women answer depending on men's replies. If a man accepts a proposal, the proposing woman accepts as well. Otherwise if he refuses, the woman shifts to the next element in her preference list and thus, makes a new proposition.

The resulting algorithm works very differently from the original version of Gale and Shapley due to asynchrony. In GSA, executions proceed in synchronous rounds. In alternating rounds, women propose in a round and in every

other round men answer. When men answer, they choose right away the best proposal in GSA. In contrary, in the proposed algorithm, they choose each time a better proposition, depending on the received proposals (scheduler). Furthermore, since GSA is centralized, if a married man accepts a better proposal, the previous marriage is canceled right away and the previous spouse can propose to another man in the next round. With an asynchronous distributed scheduler, information can be delayed and the proposed solution has to take care of that. In spite of these differences, the new asynchronous version achieves the complexity of $O(n^2)$ moves. Notice that the lower bound in $O(n^2)$ has been proven for centralized solutions and thus applies too to distributed ones.

3.1 Algorithm `Async-GSA`

Variables and Constants

Local variables at node v:

- $pref$: list of its n neighbors in preference order. The priority of the element is the rank, *i.e.*, the i^{th} element has priority i. Thus, the first element is the most preferred neighbor and its priority is 1.
- $marriage\_pref \in \mathcal{N}(v) \bigcup \texttt{Null}$: for a woman w, the node to whom w has *proposed* (and her *spouse* if additional conditions are satisfied; see below); for a man m, his *spouse* identifier. In $\mathcal{C}_{init}$, for men, the value is `Null` and for women, the first element of w's $pref$.

In the (shared) register $r_{v,u}$:
Recall that $var_{v,u}$ in register $r_{v,u}$ can be read and written by v but only read by u.

- $request_{v,u} \in \{\texttt{None}, \texttt{Proposal}, \texttt{Yes}, \texttt{No}\}$:
  - `None`: initial value of the variable (in $\mathcal{C}_{init}$) or if either v or u is `Null`.
  - `Proposal`: value available only for a woman w to *propose* to a man m.
  - `Yes`: value used by a man to *accept* a proposal or by a woman to confirm a marriage. Two nodes w and m are said to be *married* iff $request_{w,m} = request_{m,w} = \texttt{Yes}$.
  - `No`: value used by a man to *refuse* a proposal or by a woman to confirm a refusal.

Functions

- next(v): returns the element after $marriage\_pref$ in the preference list of v. Returns `Null` if $marriage\_pref$ is the last element.
- priority(v,u): returns the priority ($\in [1, n]$) of u in the preference list of v. Note that if u is evaluated to `Null`, priority(v,u) $= n + 1$.

Problem predicates

The matching $\mathcal{M}$ built by the presented algorithm is defined by so-called *married* pairs $(w, m) \in E$ such that $request_{w,m} = \texttt{Yes} \ \wedge \ request_{m,w} = \texttt{Yes} \ \wedge \ marriage\_pref_w = m \wedge \ marriage\_pref_m = w$.
The predicate defining the stable marriage is

$$\mathsf{PredSM} \equiv \ [\forall \ v \ \in \ V : \ \mathsf{Married}(v) \ \wedge \ \neg \mathsf{BlockingPair}(v)]$$

where

- $\mathsf{Married}(v) \equiv \ (request_{v,marriage\_pref_v} = \texttt{Yes}) \wedge (request_{marriage\_pref_v,v} = \texttt{Yes}) \ \wedge \ (marriage\_pref_{marriage\_pref_v} = v)$
- $\mathsf{BlockingPair}(v) \equiv \ \exists \ u \ \in \ \mathcal{N}(v) : \mathsf{priority}(v, marriage\_pref_v) > \mathsf{priority}(v, u) \wedge \mathsf{priority}(u, marriage\_pref_u) > \mathsf{priority}(u, v)$.

A pair (v, u) s.t. $\mathsf{BlockingPair}(v)$ is satisfied for a node u is a blocking pair.

Proposition 1. *A configuration C satisfies* PredSM *iff C contains a stable marriage.*

Proof. Let C be any configuration. Let us first prove by contradiction the direct implication: if PredSM is `True` in C, then C contains a stable marriage.
First, since a node v can only be married with the node $marriage\_pref_v$, v cannot be married twice. Now, let v be single. In this case, $request_{v,marriage\_pref_v} = \texttt{Yes} \wedge request_{marriage\_pref_v,v} = \texttt{Yes}$ is `False` since $request_{v,\texttt{Null}} = request_{\texttt{Null},v} = \texttt{None}$. Then, each node is married with exactly one node if *PredSM* is `True`. Furthermore, the marriage is reciprocal. Indeed, since $request_{marriage\_pref_v,v} = \texttt{Yes}$ and the predicate is `True` for the node $marriage\_pref_v$, then $marriage\_pref_{marriage\_pref_v} = v$.
Now, by contradiction, assume that v participates to a blocking pair. So, there exist node u and v, which are not married together but prefer each other to their current spouse. But PredSM is `True`, *i.e* BlockingPair(v) is `False` so that u and v do not prefer each other. These leads to a contradiction and thus, there is no blocking pair in C.
Thus, C contains a stable marriage.

Now, we prove that if a configuration C contains a stable marriage, then C satisfies PredSM. Two nodes u and v are married if $request_{v,u} = \texttt{Yes} \ \wedge \ request_{u,v} = \texttt{Yes} \ \wedge \ marriage\_pref_v = u \ \wedge \ marriage\_pref_u = v$. So, $\forall \ u \ \in \ V$, $\mathsf{Married}(v)$ is `True`. Furthermore, in a stable marriage, there is no blocking pair. Then, there is no pair (u, v) such that u prefer v to its current spouse and vice versa: the predicates $\mathsf{BlockingPair}(u)$ and $\mathsf{BlockingPair}(v)$ are false. Hence, PredSM is `True` in C.

That proves the proposition. □

Algorithm. The part of the algorithm executed by women (Algorithm 1) has 3 rules. We start by describing intuitively what those rules do.

- The rule **Propose** is executed by a woman to propose to the man in her $marriage\_pref$ pointer.

- The **Confirm** rule checks if the man $marriage\_pref$ to whom the woman has proposed, has answered positively. If he has, her register is set to `Yes`.
- The rule **Refusal_Management** is enabled if the woman's proposal has been rejected by the man $marriage\_pref$. In this case, the value `No` is set in the register and the $marriage\_pref$ pointer is set to the next man in the woman's preference list[1].

Algorithm 1 For $w \in$ WOMEN:

```
1: Propose : (* Proposes to the man pointed by marriage_pref *)
2:         { ∃ m ∈ N(w) : request_{w,m} ∉ {Proposal, Yes, No}
3:         ∧ marriage_pref = m }
4:             request_{w,m} ← Proposal
5:
6: Confirm : (* Confirms her proposal *)
7:         { ∃ m ∈ N(w) : request_{w,m} = Proposal
8:         ∧ marriage_pref = m ∧ request_{m,w} = Yes }
9:             request_{w,m} ← Yes
10:
11: Refusal_Management : (* Manages a refusal *)
12:         { ∃ m ∈ N(w) : request_{w,m} ∈ {Proposal, Yes}
13:         ∧ marriage_pref = m ∧ request_{m,w} = No}
14:             request_{w,m} ← No
15:             marriage_pref ← next(w)
```

The part of the algorithm executed by men (Algorithm 2) consists of 2 rules:

- The rule **Accept** is enabled if a woman is proposing to the man and if this woman is preferred over the actual spouse of m, *i.e.*, the woman pointed by his $marriage\_pref$ pointer. In this case, the man sets its request variable (in the shared register) to `Yes` and updates his $marriage\_pref$ pointer to the identifier of the woman.
- The role of **Refuse** is the opposite of **Accept**: if a proposal is received from a less preferred woman than his actual spouse, the man sets its request variable to `No`.

3.2 Proof of correctness and complexity

Lemma 1. *Let C_0, C_1, C_2 and C_3 be configurations. Let T_0 be a transition $C_0 \rightarrow C_1$ in which a woman w has made a proposal to m_0 in T_0. If there exists another transition T_1, $C_2 \rightarrow C_3$, in which w makes a proposal to m_1 and such that $C_1 \stackrel{*}{\rightarrow} C_2$, then, $m_0 \neq m_1$ and $priority(w, m_0) < priority(w, m_1)$.*

Proof. Let us suppose that $m_0 = m_1$. In T_0, w makes a proposal to m_0 (*i.e.*, $request_{w,m_0} \leftarrow$ `Proposal`), she has been activated for **Propose**. For the same reason, w has been activated on the link (w, m_0) for **Propose** in T_1.

[1] If the last man refuses the proposal, this rule cannot be enabled.

Algorithm 2 For $m \in$ MEN:

1: **Accept** : *(* Accepts a proposal *)*
2: $\{ \exists w \in \mathcal{N}(m) : request_{w,m} = \texttt{Proposal}$
3: $\wedge$ priority$(m,w) <$ priority$(m, marriage_pref) \}$
4: $marriage\_pref \leftarrow w$
5: $request_{m,w} \leftarrow \texttt{Yes}$
6:
7: **Refuse** : *(* Refuses a proposal *)*
8: $\{ \exists w \in \mathcal{N}(m) : request_{w,m} \in \{\texttt{Proposal}, \texttt{Yes}\} \wedge request_{m,w} \neq \texttt{No}$
9: $\wedge$ priority$(m,w) >$ priority$(m, marriage_pref) \}$
10: $request_{m,w} \leftarrow \texttt{No}$

Since in C_1 $request_{w,m_0}$ has the value `Proposal` (because of the action of **Propose**) and $request_{w,m_0} \notin \{\texttt{Yes}, \texttt{Proposal}\}$ (because $m_0 = m_1$) is satisfied in C_2, $request_{w,m_0}$ has been modified between this two configurations. Since $request_{w,m_0}$ may have only four different values, the value of $request_{w,m_0}$ can be `No` or `None` in C_2. No rule can set $request_{w,m_0}$ to `None`. Therefore, w has been activated for a **Refusal_Management** between C_1 and C_2 (the single rule that can set $request_{w,m_0}$ to `No`). At the same move, this rule has set $marriage\_pref$ to a new value by next(w). Then, the guard of **Propose** is not `True` for m_0 in C_2. Hence, the new proposal of w in C_2 is not made to m_0 but to the next(w) (let us call him m_1). Furthermore, by definition of next(w) and of the preference list, m_1 is the next element after $marriage\_pref$ in the preference list of v. That is why priority$(w,m_0) <$ priority(w,m_1). □

Corollary 1. *A woman w makes her proposals respecting her preference order such that $\forall\ m_1, m_2 \in$ WOMEN, if priority$(w, m_1) <$ priority(w, m_2) and w has made a proposal to m_2 in a configuration C, then w has already made a proposal to w_1 in a configuration C' such that $C' \rightarrow C$.*

Proof. By Lemma 1, if two proposals are made by a woman w, it is for two different men, m_0 and m_1. Furthermore, priority$(w,m_0) <$ priority(w,m_1), that is, respecting the preference order between m_0 and m_1.

Thus, all proposals (of node w) are made in her preference order.

Furthermore, let m_{00} be a man such that priority$(w,m_{00}) =$ priority$(w,m_0) - 1$. Let us prove that since w has proposed to m_0, w has already proposed to m_{00}. In $\mathcal{C}_{init}$, $marriage\_pref_w$ was pointing to the first man in her list. Since w has been activated for **Propose** to m_0, $marriage\_pref_w$ has been set to m_0 ($marriage\_pref_w = m_0$ if **Propose** is eligible). The only rule that can modify $marriage\_pref_w$ is **Refusal_Management**: this rule shifts only step by step in the list. Thus, since w is proposing to m_0, w has already proposed to m_{00}, which has refused. Furthermore, since w has proposed to m_{00}, she has already proposed to all better ranked men in her list. □

Lemma 2. *In any execution from $\mathcal{C}_{init}$, no blocking pair is created.*

Proof. Consider an execution from $\mathcal{C}_{init}$ reaching a configuration C with a blocking pair (w, m). Thus in C w is married with a man m_1, m with a woman w_1 and $\mathsf{priority}(m, w) < \mathsf{priority}(m, w_1)$ and $\mathsf{priority}(w, m) < \mathsf{priority}(w, m_1)$. But w has made proposals respecting her preference order and has already made a proposal to m (Corollary 1). Either m has refused this proposal because $\mathsf{priority}(m, w) > \mathsf{priority}(m, marriage\_pref)$, that is m was already married with a better ranked woman w_2. Or m has accepted the proposal, but, after the acceptance and before C, he has accepted a proposal from another woman w_2. Because of the condition $\mathsf{priority}(m, w_2) < \mathsf{priority}(m, marriage\_pref)$ in **Accept** guard, w_2 is better ranked than w. This case may happen several times, and, in C, w_1 is necessarily better ranked than w. Hence, (w,m) cannot be a blocking pair. □

Proposition 2. *In any execution from $\mathcal{C}_{init}$, a node v is married to its final partner after $O(n)$ of its own moves. After that, v is no longer eligible for any rule and v's variables do not change.*

Proof. Let us analyze different cases.

First, let v be a woman. From $\mathcal{C}_{init}$ by Corollary 1, a woman makes her proposals in her preference order and by Lemma 2, no blocking pair involving w is created. Since there are n nodes in each set and all nodes get married (since Gale and Shapley have proved that there always exists a stable marriage), no woman can reach the end of her list. Thus w can propose to at most n men (**Propose** rule), can be matched with at most n men (**Confirm** rule) and can be rejected by at most $n-1$ men (**Refusal_Management** rule). In each case, a man's answer takes only one move. That is, after $O(n)$ w's moves (at most $3n-1$ v's moves), a woman w is married to her final partner. In this case, she is not enabled for any rule, but as long as her spouse does not break the marriage. However, since this man is the final partner (she has already asked to all men in the worst case), that is impossible.

Now, let v be a man. v can only accept or refuse proposals. Since there is n nodes in each set and always a sable marriage, v receives at least one proposal and at most n. For each proposal, he can first accept and then refuse (because of a better proposal). Then, after at most n **Accept** and $n-1$ **Refuse** (except for the best proposal), that is $O(n)$ moves, v is married. After that, all women have already proposed to v and thus, v can only refuse the woman with whom v is married. To refuse a marriage, v must be activated for an **Accept**, to change its $marriage\_pref$ variable. But since all women have already proposed to v and v has already accepted and/or refused each, there are no more proposals to be accepted. Thus, v is no more enabled. □

Proposition 3. *Let C be a configuration reached from $\mathcal{C}_{init}$ where no node is enabled (i.e. C is terminal). The set of edges $\{(w, m) \in E : request_{w,m} = request_{m,w} = \texttt{Yes}\}$ is a stable matching.*

Proof. Let C be a configuration with no stable marriage where no node is enabled. The two cases are: 1. at least two nodes are single (a man and a woman), 2. there exists at least one blocking pair. The case 2 is impossible by Lemma 2.

Thus, let us consider case 1. Consider a single women w. Since in C no node is enabled, she can not be activated for **Propose** and she cannot be waiting for an answer (an other node enabled to confirm or refuse). A possibility is that she has reached the end of her list. But since there are n men and n women and each node can be married with only one node ($marriage\_pref$ can store only one value), some man m is also single. Thus, $marriage\_pref_w = marriage\_pref_m = \texttt{Null}$ and then $\mathsf{priority}(m, w) > \mathsf{priority}(m, marriage\_pref_m) \wedge \mathsf{priority}(w, m) > \mathsf{priority}(w, marriage\_pref_w)$. Thus there is a blocking pair, contradictory to Lemma 2. This leads to a contradiction. Then in a configuration where no node is enabled, no node can be single.

Hence, if no node is enabled in C, all nodes are married and there is no blocking pair, whence a stable marriage. □

Theorem 1. *From $\mathcal{C}_{init}$, after at most $O(n^2)$ moves, a terminal configuration with a stable marriage is reached.*

Proof. Since there are n women and since, by Prop. 2, each woman needs $O(n)$ moves before getting married to its final partner, all women are irrevocably married after $O(n^2)$ moves. Since there are exactly n men and since a woman cannot be married to more than one man, after $O(n^2)$ moves all men are married too to their final partners. After that, by Prop. 2, there are no more enabled nodes. Thus a terminal configuration is reached and by Prop. 3, it contains a stable marriage. □

Complexity in term of rounds. The definition of round captures the execution rate of the slowest processor in any computation. Since there is at least one move in each round, the move complexity is also an upper bound for the round complexity. Thus, the final configuration is reached in at most $O(n^2)$ rounds. Notice that this is also a tight bound $\Theta(n^2)$, because there is an execution taking at least $\Theta(n^2)$ moves.

The given algorithm is an asynchronous distributed version of the Gale and Shapley algorithm. It must be executed from an initial configuration $\mathcal{C}_{init}$. From an arbitrary configuration, the algorithm does not always reach a terminal configuration with a stable marriage: blocking pairs may continue to exist or nodes may remain single. We will prove that it is possible to detect locally these problems. Indeed, the presence of a blocking pair can be detected by the two concerned nodes that can check whether they prefer each other. A possibility could be to resolve each blocking pair by matching the two nodes. But this solution can lead to cycles [21] and never reach a terminal configuration (because blocking pairs can be created while others are resolved). Thus, we use another technique, based upon local checking and global reset [4]. The next section describes this technique.

4 Self-stabilizing stable marriage by local detection and global correction

It was proven in [4] that, if an initialized solution satisfies some specific properties, it can be transformed in a self-stabilizing solution. Although [4] assumes the message passing model, the transformation applies to the link register model. Indeed in [4] channel capacity is 1 and is equivalent to a register in read/write atomicity.

For the transformation to be correct, the non self-stabilizing algorithm has to be *locally checkable*, *i.e.*, nodes can locally detect if a configuration is incorrect. Correct configurations are those reached by an execution starting from the initial configuration. Checking is made periodically by a node, on the couple of its own state and the state of one of its neighbors. If an inaccuracy is detected, a global reset is launched, setting each variable to a predefined value. Then the algorithm behaves as if it had been started in an initial configuration and reaches a terminal configuration with a stable marriage.

Thus, the issue is to design a locally checkable solution. Roughly, checking that the edge (m, w) is not a blocking pair can be made locally: w gives to m its priority and the priority of its current spouse. With this information, m is able to detect a blocking pair but also an incoherence in the variables. We prove it in the subsection 4.1. Notice that the exchange of information between w and m is limited and respects the privacy: preference lists are not communicated.

4.1 Local checkability

The definition is adapted from [4]. We define the local predicate $\mathsf{LP}_{m,w}$ on the link (m, w), *i.e.*, a predicate on the registers $r_{m,w}$, $r_{w,m}$ and the local variables m and w. We define the global predicate Π on a configuration, *i.e.*, a predicate on all registers and local variables.

Definition 2 (Local Checkability)**.** *A solution* `Alg` *to a problem is locally checkable for* Π *iff the following conditions hold.*

1. *There exists a set of local predicates* $\mathsf{LP}_{m,w}$*, for each man* m *and each woman* w *such that*
$$\Pi = \bigwedge_{\forall (m,w) \in E} \mathsf{LP}_{m,w}.$$
2. *There exists a configuration of* `Alg` *satisfying* $\mathsf{LP}_{m,w}$ *for all* $\mathsf{LP}_{m,w}$*.*
3. *Each* $\mathsf{LP}_{m,w}$ *is such that, if C is a configuration satisfying* $\mathsf{LP}_{m,w}$ *and C* $\rightarrow$ *C' is a transition, then C' satisfies also* $\mathsf{LP}_{m,w}$*.*

Now we prove that `Async-GSA` is locally checkable for Π and define the $\mathsf{LP}_{m,w}$ as follows.

Local predicates. The local predicate $\mathsf{LP}_{m,w}$ must detect any deviation in the execution of `Async-GSA`. Hence, it must describe the local views in all reachable configurations. That is why we build it from the guarded rules of `Async-GSA`. Let us describe all these configurations for any (m, w).

First, consider a configuration C, where $request_{w,m}$ and $request_{m,w} = \texttt{None}$, and all configurations reached from it with no rule applied on (m, w). In these latter configurations, $request_{w,m}$ and $request_{m,w} = \texttt{None}$ have not been changed but $marriage\_pref_w$ and $marriage\_pref_m$ may have been updated by rules applied on other links. Let $\mathsf{P}^1_{m,w}$ be the predicate describing the state of (m, w) in these configurations.

$$\mathsf{P}^1_{m,w} \equiv request_{w,m} = request_{m,w} = \texttt{None}$$

Now, let $\mathsf{P}^2_{m,w}$ be the predicate describing a configuration in which a proposal has been made by woman w. Proposals are made in a configuration satisfying $\mathsf{P}^1_{m,w}$ with **Propose**. This rule sets $request_{w,m}$ to the value `Proposal` and $marriage\_pref_w$ to m.

$$\mathsf{P}^2_{m,w} \equiv request_{w,m} = \texttt{Proposal} \wedge marriage\_pref_w = m \\ \wedge request_{m,w} = \texttt{None}$$

From a configuration satisfying $\mathsf{P}^2_{m,w}$, if m is activated for **Refuse** to refuse w's proposal, the configuration is in $\mathsf{P}^3_{m,w}$. **Refuse** is eligible if w has a worse priority than $marriage\_pref_m$ and sets $request_{m,w}$ to `No`.

$$\mathsf{P}^3_{m,w} \equiv request_{w,m} = \texttt{Proposal} \wedge marriage\_pref_w = m \\ \wedge request_{m,w} = \texttt{No} \wedge \mathsf{priority}(m,w) > \mathsf{priority}(m, marriage\_pref_m)$$

The other possibility is that, from a configuration satisfying $\mathsf{P}^2_{m,w}$, an acceptance is made by m with **Accept**. Thus, $\mathsf{P}^4_{m,w}$ is the predicate describing a configuration in which a proposal has been made by w to m and m has accepted. The predicate checks the priority of $marriage\_pref_w$: after **Accept**, either $marriage\_pref_m$ points to w or, if m has accepted a new better proposal, to a better ranked woman. Thus, the priority of $marriage\_pref_m$ is better than w.

$$\mathsf{P}^4_{m,w} \equiv request_{w,m} = \texttt{Proposal} \wedge request_{m,w} = \texttt{Yes} \\ \wedge marriage\_pref_w = m \wedge \mathsf{priority}(m,w) \geq \mathsf{priority}(m, marriage\_pref_m)$$

$\mathsf{P}^5_{m,w}$ is the predicate describing a configuration in which both nodes have accepted: they are married. This configuration is obtained from a configuration satisfying $\mathsf{P}^4_{m,w}$ after a transition with **Confirm**. For the same reason than for $\mathsf{P}^4_{m,w}$, the priority of $marriage\_pref_m$ is checked.

$$\mathsf{P}^5_{m,w} \equiv request_{w,m} = \texttt{Yes} \wedge request_{m,w} = \texttt{Yes} \\ \wedge \mathsf{priority}(m,w) \geq \mathsf{priority}(m, marriage\_pref_m)$$

Predicate $\mathsf{P}^6_{m,w}$ describes a configuration in which w and m were married but m has refused after having accepted. This configuration is obtained from a configuration satisfying $\mathsf{P}^6_{m,w}$ after a transition with the men's **Refuse**. **Refuse** is eligible if w has a worse priority than $marriage\_pref_m$ and sets $request_{m,w}$ to `No`.

$$\mathsf{P}^6_{m,w} \equiv request_{w,m} = \texttt{Yes} \wedge marriage\_pref_w = m \wedge request_{m,w} = \texttt{No} \\ \wedge \mathsf{priority}(m,w) > \mathsf{priority}(m,marriage\_pref_m)$$

The last configuration case is reached from a configuration satisfying $\mathsf{P}^3_{m,w}$ or $\mathsf{P}^6_{m,w}$ by the transition **Refusal_Management** of w's. After the transition, the configuration is in $\mathsf{P}^7_{m,w}$, the predicate describing a configuration in which w and m have both refused to be married together.

$$\mathsf{P}^7_{m,w} \equiv request_{w,m} = \texttt{No} \wedge marriage\_pref_w \neq m \\ \wedge request_{m,w} = \texttt{No} \wedge \mathsf{priority}(m,w) > \mathsf{priority}(m,marriage\_pref_m)$$

At the end, the predicate $\mathsf{P}^{BP}_{m,w}$ detects if the subsystem contains a blocking pair. There exists a blocking pair if both $marriage\_pref$ variables are not pointing to each other but nodes prefer each other to their actual value of $marriage\_pref$, for all values of $request$.

$$\mathsf{P}^{BP}_{m,w} \equiv \mathsf{priority}(m,w) < \mathsf{priority}(m,marriage\_pref_m) \\ \wedge \mathsf{priority}(w,m) < \mathsf{priority}(w,marriage\_pref_w)$$

Thus, the local predicate checked by m is the following.

$$\mathsf{LP}_{m,w} \equiv (\mathsf{P}^1_{m,w} \vee \mathsf{P}^2_{m,w} \vee \mathsf{P}^3_{m,w} \vee \mathsf{P}^4_{m,w} \vee \mathsf{P}^5_{m,w} \vee \mathsf{P}^6_{m,w} \vee \mathsf{P}^7_{m,w}) \wedge \neg \mathsf{P}^{BP}_{m,w}$$

Local checkability. Now we prove the local checkability of `Async-GSA`.

First, let us consider the condition 3 in Def. 2, *i.e.* that if C is a configuration satisfying $LP_{m,w}$, reachable from the initial configuration and $C \rightarrow C$' is a transition, then C' satisfies also $LP_{m,w}$. Since $\mathsf{P}^1_{m,w}$, $\mathsf{P}^2_{m,w}$, $\mathsf{P}^3_{m,w}$, $\mathsf{P}^4_{m,w}$, $\mathsf{P}^5_{m,w}$, $\mathsf{P}^6_{m,w}$ and $\mathsf{P}^7_{m,w}$ were constructed directly from guards and actions of rules, this is the case. Thus, since in $\mathsf{LP}_{m,w}$ the negation of $\mathsf{P}^{BP}_{m,w}$ is used, we have to prove that if $\mathsf{P}^{BP}_{m,w}$ is `False` in a configuration, it remains `False` in the following configurations.

Lemma 3. *The predicate $\neg\mathsf{P}^{BP}_{m,w}$ satisfies point 3 of the Definition 2, i.e., for any transition C$\rightarrow$ C', if C does not satisfy $\mathsf{P}^{BP}_{m,w}$, then neither does C'.*

Proof. Assume that C does not satisfy $\mathsf{P}^{BP}_{m,w}$. We prove below that C' does not satisfy it neither.

If m is activated for **Accept**, $\mathsf{priority}(m,w) < \mathsf{priority}(m,marriage\_pref_m)$ is `True` in C. But, since $\mathsf{P}^{BP}_{m,w}$ is not satisfied in C, we have $\mathsf{priority}(w,m) > \mathsf{priority}(w,marriage\_pref_w)$ in C and this is still `True` in C' (**Accept** does not change $marriage\_pref_w$). Thus, $\mathsf{P}^{BP}_{m,w}$ is also `False` in C'.

Rules **Propose**, **Confirm** and **Refuse** do not change the value of $mariage\_pref$ of w and m. Thus, in C', $\mathsf{P}^{BP}_{m,w}$ is still `False`.
If w is activated for **Refusal_Management**, in C', $marriage\_pref_w$ is shifted to the next element in the preference list of w. This rule is eligible only if $request_{m,w} = \texttt{No}$. Thus, in the previous transition, m has set her variable to `No` with the **Refuse** rule, *i.e.* $\mathsf{priority}(m,w) > \mathsf{priority}(m,marriage\_pref_m)$ is `False` in C and is still `True` in C'. Hence, $\mathsf{P}^{BP}_{m,w}$ is still `False` in C'.

So, the predicate $\neg\mathsf{P}^{BP}_{m,w}$ satisfies the condition. □

Now, let us prove that `Async-GSA` is locally checkable.

Theorem 2. *Let* $\Pi = \bigwedge_{\forall(m,w)\in E} LP_{m,w}$. `Async-GSA` *is locally checkable for* Π.

Proof. There are three conditions in Def. 2 to satisfy.

First, the condition 1 is satisfied by the definition of Π and $\mathsf{LP}_{m,w}$.

Second, the configuration C in which all variables are set to their initial values satisfies $\mathsf{LP}_{m,w}$ because $request_{m,w} = request_{w,m} = \texttt{None}$.

Finally, by Lemma 3 and construction of other predicates, the condition 3 of the locally checkable property is satisfied.

This concludes the proof. □

Now `Async-GSA` being locally checkable, a man can detect, using the local snapshot detailed in [4], whether the state does not satisfy $\mathsf{LP}_{m,w}$. In this case, m launches a reset.

Upon request, the reset sets all variables of the system to the values of the starting configuration of `Asyn-GSA`. To propagate this reset, a spanning tree is built in the following way.

4.2 Reset tree construction

The structural properties of the bipartite communication graph are used for building a tree of depth 2. The woman with the minimum identifier W_{min} is the root of the tree. The children of W_{min} are all the men. The other women are at distance two from W_{min} and are descendants of the man with the minimum identifier M_{min}. Thus, the tree has a depth of 2. Since each node holds the identifiers of the other subset in its preference list, W_{min} learns that she is the root from the men. Following the same process, M_{min} learns from the women that he has the minimum identifier.

It is easy to see that roughly, an activation of all the men followed by an activation of all the women is repeated twice, the tree is stabilized. Notice however that this may take $O(n^2)$ moves. This is because nodes enabled for the tree construction can be deferred from being chosen by the scheduler, until there are other enabled nodes for executing the rules of `Async-GSA`. Assuming the correctness of the detection module, this can take at most $O(n^2)$ moves (Theorem 1).

4.3 Composition and analysis

One can see that the overall complexity of the algorithm is of $O(n^2)$ moves. In the worst case, after the tree has been built (in at most $O(n^2)$ moves), an execution is divided in three parts: an initial part in which a reset is enabled but not triggered, a second part during which a reset is performed and a third part, which corresponds to an execution with initialization (of `Async-GSA`). We discuss upper bounds for each of these parts. Without loss of generality, we assume that the guards of a node are evaluated in the following order: first the spanning tree construction, second the rules involved in the reset, then the rules checking an incoherence or a blocking pair and at the end the rules of `Async-GSA`.

For the last part, Theorem 1 gives the $O(n^2)$ upper bound in terms of moves and this is an upper bound in term of rounds. Now, consider the first part. There are some incoherent nodes or nodes involved in a blocking pair in the starting configuration. The other nodes simply execute rules of the algorithms 1 and 2. The longest execution segment of this part is obtained when the unfair scheduler chooses to ignore the incorrect nodes (from executing the rules of the incoherence detection). This may take at most $O(n^2)$ moves: after a partial stable marriage has been built with the correct nodes (using Algorithms 1 and 2), these nodes are no more eligible, at least because no woman can make a new proposition, as the end of her preference list has been reached. Then, an incorrect node must be activated, triggering a reset. The task of building a partial stable marriage takes $O(n^2)$, still from Theorem 1.

At the end, for the reset part, we adopt the algorithm from [3] (it satisfies the conditions required by [4]). This algorithm can be viewed as proceeding in "waves" (of broadcast and convergecast) propagated over a tree and coordinated by the root, in a similar way as the propagation of information (PIF) can be used for a reset over a tree (see e.g. [7]). Such reset can be decomposed in three subparts: a) a reset request launched towards the root, b) a "freezing" wave from the root to the leaves, initializing the `Async-GSA` variables (following by a feedback to the root), and then c) an "unfreezing" wave from the root to all the nodes, launching the `Async-GSA`. These waves are diffused on the constructed tree, which is of height 2. Each wave takes $O(n)$ moves. Furthermore, the reset has a delay before being operational since reset variables can be incoherent. Even if each node initializes a wrong reset, that takes less than $n \times O(n) = O(n^2)$ moves. After a reset has been accomplished, variables are set to their initial values and `Async-GSA` can start. Furthermore, this reset has a round complexity of $O(d)$ [4], *i.e.*, has a constant round complexity of $O(2)$.

This justifies the overall complexity of $O(n^2)$ moves and rounds.

References

[1] H. Ackermann, P. W. Goldberg, V. S. Mirrokni, H. Röglin, and B. Vöcking. Uncoordinated two-sided matching markets. *SIAM J. Comput.*, 40(1):92–

106, 2011.

[2] N. Amira, R. Giladi, and Z. Lotker. Distributed weighted stable marriage problem. In *SIROCCO 2010*, pages 29–40, 2010.

[3] A. Arora and M. Gouda. Distributed reset. *IEEE Transactions on Computers*, 43(9):1026–1038, Sep 1994.

[4] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev. Self-stabilization by local checking and global reset. In *Distributed Algorithms*, pages 326–339, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[5] R. Bai, J. Li, J. A. D. Atkin, and G. Kendall. A novel approach to independent taxi scheduling problem based on stable matching. *Journal of the Operational Research Society*, 65(10):1501–1510, Oct 2014.

[6] I. Brito and P. Meseguer. Distributed stable marriage problem. In *6th Workshop on Distributed Constraint Reasoning at IJCAI*, volume 5, pages 135–147, 2005.

[7] A. Bui, A. K. Datta, F. Petit, and V. Villain. State-optimal snap-stabilizing PIF in tree networks. In *Proceedings 19th IEEE International Conference on Distributed Computing Systems*, pages 78–85, 1999.

[8] J.-H. Chou and C.-J. Lu. Communication requirements for stable marriages. In *Algorithms and Complexity: 7th International Conference, CIAC 2010, Rome, Italy, May 26-28, 2010. Proceedings*, pages 371–382, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[9] S. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input/output-queued switch. *IEEE Journal on Selected Areas in Communications*, 17(6):1030–1039, 1999.

[10] A. Cournier, A. K. Datta, F. Petit, and V. Villain. Snap-stabilizing PIF algorithm in arbitrary networks. In *ICDCS*, pages 199–206, 2002.

[11] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distrib. Comput.*, 7(1):3–16, November 1993.

[12] S. Dolev, A. Israeli, and S. Moran. Uniform dynamic self-stabilizing leader election. *IEEE Trans. Parallel Distrib. Syst.*, 8(4):424–440, 1997.

[13] P. Floréen, P. Kaski, V. Polishchuk, and J. Suomela. Almost stable matchings by truncating the gale-shapley algorithm. *Algorithmica*, 58(1):102–118, 2010.

[14] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 120(5):386–391, 1962.

[15] P. Golle. A private stable matching algorithm. In *FC*, pages 65–80, 2006.

[16] Y. A. Gonczarowski, N. Nisan, R. Ostrovsky, and W. Rosenbaum. A stable marriage requires communication. In *SODA '15*, pages 1003–1017, 2015.

[17] D. Gusfield and R. W. Irving. *The Stable marriage problem - structure and algorithms*. Foundations of computing series. MIT Press, 1989.

[18] P. Khanchandani and R. Wattenhofer. Distributed stable matching with similar preference lists. In *OPODIS*, pages 12:1–12:16, 2016.

[19] G. Kim and W. Lee. Stable matching with ties for cloud-assisted smart tv services. In *ICCE*, pages 558–559, 2014.

[20] A. Kipnis and B. Patt-Shamir. A note on distributed stable matching. In *ICDCS*, pages 466–473, 2009.

[21] D. E. Knuth. *Mariages stables et leurs relations avec d'autres problemes combinatoires*. Les Presses de l'Université de Montréal, 1976.

[22] M. Kuemmel, F. Busch, and D. Z.W. Wang. Taxi dispatching and stable marriage. 12 2016.

[23] L. and C. Johnen. Relationships between communication models in networks using atomic registers. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, 2006.

[24] M. Laveau, G. Manoussakis, J. Beauquier, T. Bernard, J. Burman, J. Cohen, and L. Pilard. Self-stabilizing distributed stable marriage. In *Stabilization, Safety, and Security of Distributed Systems*. Springer International Publishing, 2017.

[25] B. M. Maggs and R. K. Sitaraman. Algorithmic nuggets in content delivery. *Computer Communication Review*, 45(3):52–66, 2015.

[26] C. Ng and D. S. Hirschberg. Lower bounds for the stable marriage problem and its variants. *SIAM Journal on Computing*, 19(1):71–77, 1990.

[27] R. Ostrovsky and W. Rosenbaum. Fast distributed almost stable matchings. In *PODC'15*, pages 101–108, New York, NY, USA, 2015. ACM.

[28] I. Segal. The communication requirements of social choice rules and supporting budget sets. *Journal of Economic Theory*, 136(1):341 – 378, 2007.

[29] H. Xu and B. Li. Egalitarian stable matching for VM migration in cloud computing. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2011.

[30] A. C.-C. Yao. Some complexity questions related to distributive computing(preliminary report). In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79. ACM, 1979.