Mariage stable asynchrone auto-stabilisant

Marie Laveau

Mariage stable asynchrone auto-stabilisant

Marie Laveau [1]

[1] *LRI, Univ. Paris-Sud, Université Paris-Saclay, 91400, Orsay, France, {jb, laveau}@lri.fr*

Le problème du *mariage stable* est un problème d'appariement d'éléments de deux ensembles distincts. L'appariement se construit selon les préférences des éléments afin de satisfaire leurs besoins "égoïstes". Ce problème a été initialement étudié dans un contexte centralisé, puis diverses solutions réparties ont été présentées dans un cadre synchrone. Nous proposons pour la première fois une solution dans un contexte *distribué*, *asynchrone* et *auto-stabilisant*. Nous présentons la preuve de la correction de l'algorithme proposé et analysons la complexité.

Keywords : algorithmique distribuée, mariage stable, asynchronisme, auto-stabilisation

1 Introduction

Les problèmes d'appariement sont classiques et très étudiés. Nous nous intéressons à une variante du problème connue sous le nom de mariage stable, introduit par Gale et Shapley [GS62], qui a de nombreuses applications. Par exemple, lorsqu'il a été présenté pour la première fois, l'objectif était l'appariement de places d'internat dans les hôpitaux avec les internes en médecine. Plus tard, il a été utilisé pour représenter des marchés avec des clients et des vendeurs ou des ensembles de clients et de serveurs pour des problèmes de migration de machines virtuelles dans le Cloud [XL11a]. La différence entre le mariage stable et un appariement classique (comme le maximal matching) est l'existence, pour chaque processus, d'une liste de préférences. La présentation de Gale et Shapley utilise un graphe biparti avec d'un côté les hommes et de l'autre les femmes. Chacun a une liste de préférences classant les conjoints potentiels. Le mariage est stable s'il ne contient pas de paire bloquante (un homme et une femme non mariés ensemble qui se préfèrent mutuellement à leur conjoint respectif).

Gale et Shapley ont proposé une solution centralisée qui résout le problème en $O(n^2)$ étapes. Nous proposons une solution distribuée, auto-stabilisante et asynchrone. Rappelons qu'un système est dit auto-stabilisant, si quelque soit sa configuration initiale, il converge vers un ensemble de configurations vérifiant les spécifications du problème qu'il est censé résoudre (*cf.* [Dij74]). De nombreux travaux ont été menés dans le cadre synchrone (e.g. [OR15], [Mat08], [KPS09]). Plus rarement, des travaux ont été menés dans le cadre asynchrone [XL11b], [BM05a].

L'algorithme de Gale et Shapley n'est pas auto-stabilisant. Initialisé avec des valeurs correctes, l'algorithme réduit les paires bloquantes, mais mal initialisé il peut, de manière cyclique, recréer les paires bloquantes qu'il avait éliminées auparavant [Rapport Nř]. Nous avons ainsi mis en place un mécanisme de détection local qui, lorsqu'il est déclenché, provoque une réinitialisation globale du système dans une configuration ne menant pas à l'obtention de paire bloquante. La réinitialisation du système est réalisée par la solution de Dolev [Dol00] sur un arbre dont nous détaillerons la construction.

Notre technique diffère de la technique générale de Awerbuch [APSVD94], qui s'applique aux problèmes qui peuvent être corrigés localement, ce qui n'est pas le cas du mariage stable. La transformation très générale de Katz et Perry[KP90], consistant à prendre régulièrement des instantanés du système et à faire des corrections globales en cas de corruption s'applique, elle, au problème. Mais à sa différence, notre détection est locale et donc beaucoup moins coûteuse en terme de messages. Pour conclure, signalons que l'extension du reset de Dolev aux systèmes à passage de messages nécessite que les liens soient FIFO, mais que cette condition peut être obtenue en utilisant le bit alterné auto-stabilisant d'Afek et Brown [AB93].

2 Modèle et définitions

Modèle Le modèle considéré ici est un système composé de N agents ayant chacun un ID (identificateur) unique et non corruptible. Ils sont répartis en deux sous-groupes de même taille n. Le graphe de commu-

nications est un graphe biparti complet. Chaque nœud connaît ses voisins dans le graphe (i.e. nœuds qui appartiennent à l'autre sous-groupe) par leur identificateur ainsi que par le canal qui les relie. Pour cela, ils sont stockés dans une liste de préférences non corruptible. Les autres variables sont, quant à elles corruptibles. Ce système utilise un modèle asynchrone classique à passage de messages, sans perte de message. De plus, les canaux sont FIFO et bidirectionnels. Sans perte de généralité, nous pouvons dire que le nombre de messages en transit dans chaque sens et au même moment est borné à 1.

Auto-stabilisation Un algorithme auto-stabilisant permet de résoudre les problèmes dans un contexte de défaillances transitoires. En effet, à partir d'une configuration quelconque qui peut être due à une défaillance transitoire, l'algorithme atteint nécessairement une configuration légitime qui répond à la spécification. Une *défaillance transitoire* altère le contenu des canaux de communication ou la mémoire des processus de manière arbitraire.

Soit $\mathcal{C}_{\mathcal{L}}$ l'ensemble des *configurations légitimes*. Le système $\mathcal{S}$ est dit *auto-stabilisant* pour la spécification $\mathcal{P}$ si et seulement si il existe un sous-ensemble $\mathcal{C}_{\mathcal{L}}$ de l'ensemble $\mathcal{C}$ des configurations de $\mathcal{S}$ tel que toute exécution de $\mathcal{S}$ dont la configuration initiale appartient à $\mathcal{C}_{\mathcal{L}}$ vérifie $\mathcal{P}$. De plus, toute exécution de $\mathcal{S}$ qui vérifie les spécifications du problème contient au moins une configuration appartenant à $\mathcal{C}_{\mathcal{L}}$.

Mariage stable Le problème du mariage stable est posé en théorie des graphes sur des graphes bipartis $K_{n,n}$. Le but est d'associer deux à deux les nœuds de chacune des parties, tout en prenant en compte les préférences de chaque nœud .

Étant donné un graphe biparti complet non orienté $G = (V = A \bigcup B,\ E) = K_{n,n}$ où A et B sont les deux parties de taille n du graphes G et $E = \{(i, j) : i \in A, j \in B\}$, l'ensemble des arêtes du graphe. Chaque nœud $v \in A$ a un identificateur, ainsi qu'une liste de préférence pour les nœuds de B et réciproquement. Il s'agit d'associer deux à deux les éléments des ensembles A et B sans qu'il y ait deux éléments α et β (respectivement dans A et B), chacun appariés à d'autres élément, qui se préfèrent mutuellement. Dans ce cas, la paire (α, β) est une *paire bloquante* et le mariage est *non stable*.

Gale et Shapley ont défini ce problème en prenant un ensemble d'hommes et un ensemble de femmes. Nous parlerons donc de nœud sans référence spécifique à un sous-ensemble et sinon d'hommes et de femmes. Leur algorithme [GS62] fonctionne par *acceptation différée* : chaque élément des deux sous-ensembles (homme ou femme) connaît sa préférence courante. Au début, les hommes envoient simultanément une invitation à la femme la mieux classée dans leurs liste. Les femmes répondent positivement à la meilleure proposition reçue d'après leurs préférences. Aux autres hommes, elles envoient un refus. Celles qui ne reçoivent pas de proposition attendent. Ce processus est réitéré jusqu'à ce que tous les appariements soient faits. Une femme qui reçoit une meilleure proposition romp son appariement et prévient son partenaire. Celui-ci adresse alors une demande à la femme suivante dans sa liste. Cet algorithme obtient une solution en $O(n^2)$ étapes. Gale et Shapley ont prouvé par ailleurs qu'il existe toujours un mariage stable pour un graphe biparti complet.

3 Solution auto-stabilisante du mariage stable

Description Nous construisons le mariage stable par envoi de propositions, à la manière de Gale et Shapley, mais de manière auto-stabilisante. En effet, dans un cadre asynchrone, l'algorithme originel ne s'adapte pas aux possibilités des fautes. Un homme pourrait croire, après corruption de mémoire, qu'il est marié avec une femme alors que ce n'est pas le cas. Pour y éviter ceci, une fois marié, l'homme vérifie la présence de paire bloquante auprès des femmes mieux classées que sa femme actuelle. S'il détecte une paire bloquante, il change de mariage en informant son 'ex-femme'. Si un homme arrive à la fin de sa liste sans avoir trouvé d'appariement, il ne peut revenir au début de la liste pour continuer sa recherche. En effet, comme il y a un nombre égal d'hommes et de femmes et que Gale et Shapley ont prouvé qu'il y a forcément un mariage stable, cela signifie qu'il y a une erreur. Á ce moment là, si un ordre de message précis est répété, cet homme pourrait boucler infiniment sur sa liste. C'est pourquoi il enclenche un *reset* pour réinitialiser tous les nœuds . Il est possible de composer notre algorithme de mariage avec un reset auto-stabilisant en utilisant la technique de composition équitable [DIM93].

Le reset est mis en œuvre en utilisant une structure d'arbre : les messages sont routés sur ses arêtes. La racine de l'arbre est la femme avec l'identificateur minimum, *F_min*. Les nœuds à distance 1 de la racine sont tous les hommes. Les autres femmes sont à distance 2 de la racine et sont les descendantes de l'homme

avec l'identifiant minimum H_{min}. Comme les nœuds ne disposent que de la listes des identifiants de l'autre sous ensemble, *F_min* va apprendre qu'elle est racine grâce aux hommes qui vont envoyer de manière répétée un message d'information `min`.Aux autres femmes, ils vont envoyer un message `no_min`. De manière similaire, H_{min} va apprendre qu'il a l'identifiant minimum grâce aux femmes. Ainsi, si un homme veut envoyer un message à une femme, le message parcourt l'arbre : l'acheminement d'un message utilise donc au plus trois messages. Les nœuds utilisent les primitives Envoyer(*msg*, *v*) et Réception(*msg*, *v*) qui réalisent routage des messages de la source au destinataire final via les arêtes de l'arbre.

Algorithme Les hommes envoient des `Propositions` aux femmes dans l'ordre croissant de préférence (Algo 1, ligne 3). Celles-ci répondent (Algo 2) avec des messages `Oui` (Ligne 3 et 5) ou `Non` (Ligne 9, en fonction des autres propositions reçues et du classement de chaque homme dans leur liste. Quant une femme refuse, l'homme demande à la suivante dans la liste de préférences (Ligne 16). De plus, si un homme reçoit un `Oui`, il vérifie que c'est bien le meilleur choix (Ligne 9) et enregistre l'identifiant (Ligne 12) avant de recommencer au début de la liste pour vérifier l'existence de paires bloquantes en remettant le *pref_courante* à 1 (Ligne 13). L'identifiant du nœud associé est stocké dans la variable *mariage_actuel*. Si un homme atteint la fin de sa liste sans avoir trouvé de mariage, il enclenche un reset. (Algo 1, ligne 5). Cela signifie qu'une erreur est survenue, étant donné qu'il y a autant d'hommes que de femmes. Le reset met à n le $pref\_courante$ et à $\varnothing$ la variable $mariage\_actuel$.

Variables :

mariage_actuel : identifiant /* *Id du nœud avec lequel il est marié, réinitialisé à* $\varnothing$ */
invoke : Booléen /* *Variable du module de reset, indique si un problème a été détecté ou non* */
reset : Booléen /* *Variable du module de reset, indique si un reset est en cours* */
Pref : liste non corruptible de taille n d'identifiants /* *Contient tous les identifiants des voisins dans l'ordre de préférence décroissant* */
pref_courante : entier de 1 à n /* *Contient l'index du nœud courant* */

Fonctions et prédicat :

index(*v* : identifiant) → entier /* retourne l'index du noeud v dans la liste **Pref** */
Reset() : Déclenche le reset de Dolev en changeant la variable *invoke* à *true*
Envoie(*msg*, *v*) : envoie *msg* ≡ [`Proposition`, `Non`, `Oui`] à *v* en utilisant l'arbre pour transmettre.
PReset : *reset* = true ∧ *invoke* = true, où *invoke* et *reset* sont des variables du module de reset

Reset Nous utilisons ici le reset auto-stabilisant présenté par Dolev [Dol00]. Celui-ci fonctionne sur un arbre non orienté. Nous utilisons donc l'arbre précédemment décrit pour diffuser le reset. La solution présentée par Dolev est à lecture de registre et non passage de message. Il est cependant possible d'adapter cette solution au modèle à passage de message à l'aide d'une des méthodes présentées par Varghese [Var94] ou Katz et Perry [KP90]. Dans cet algorithme, chaque nœud a une variable *reset* qui stocke si un reset est en cours et une variable *invoke* pour annoncer qu'il y a besoin d'un reset. Ils vont envoyer de manière répétée leurs valeurs à leurs voisins pour permettre l'échange d'information et la diffusion du reset. Si un homme détecte un problème dans l'algorithme de mariage et donc provoque un reset, il va donc modifier sa variable *invoke* à *true* pour faire remonter à la racine la demande de reset. Cette valeur va remonter à la racine qui va enclencher le reset. Les variables *reset* et *invoke* sont lues par l'algorithme de mariage pour savoir si on est en phase de reset ou non : si c'est le cas, les nœuds vont être gelés et ne plus pouvoir communiquer pour le mariage. Il n'y a plus de messages qui pourrait gêner la stabilisation après le reset. D'autre part, le reset permet de mettre la variable *mariage_actuel* à $\varnothing$ et le *pref_courante* à 1, ce qui correspond à l'état légitime du début de l'algorithme originel d'acceptation différée. Étant donné qu'il utilise la structure d'arbre et que les canaux sont FIFO, le reset vide les canaux des messages du mariage précédent ou des messages arbitrairement initialisés.

Algorithme 1 Hommes

```
1:  ¬ PReset →
2:      si pref_courante ≠ 0 alors
3:          Envoie(Proposition, Pref[pref_courante])    // Détection des paires bloquantes
4:      sinon    // Si l'homme arrive au bout de sa liste ⇒ Problème
5:          Reset()
6:
7:  Sur réception de (msg, v) et ¬ PReset →
8:      si msg = Oui alors    // La femme v accepte/confirme l'appariement
9:          si index(v) ≤ index(mariage_actuel) alors
10:             si v ≠ mariage_actuel alors
11:                 Envoie(Non, mariage_actuel)
12:             mariage_actuel ← v
13:             pref_courante ← n
14:         sinon si index(v) > index(mariage_actuel) alors
15:             Envoie(Non, v)
16:     sinon si msg = Non alors    // La femme v refuse l'appariement
17:         si v = mariage_actuel alors
18:             mariage_actuel ← ∅
19:         sinon
20:             pref_courante ← index(v) + 1
```

Algorithme 2 Femmes

```
1:  Sur réception de (msg, v) et ¬ PReset →
2:      si msg = Proposition alors
3:          si mariage_actuel = v alors    // Accepte l'appariement ou valide la vérification
4:              Envoie(Oui, v)
5:          sinon si index(v) < index(mariage_actuel) alors    // Construction du mariage
6:              Envoie(Non, mariage_actuel)
7:              Envoie(Oui, v)
8:              mariage_actuel ← v
9:          sinon    // Refuse l'homme
10:             Envoie(Non, v)
11:     sinon si msg = Non alors    // Si l'homme a un meilleur appariement, on remet mariage_actuel à ∅
12:         si mariage_actuel = v alors
13:             mariage_actuel ← ∅
```

4 Preuve et complexité

Preuve Nous supposons que, lorsque le système converge, aucune nouvelle corruption ne peux survenir. Nous partons donc d'une configuration quelconque.

Lemme 1 *Il n'y aura pas un nombre infini de reset enclenché.*

Si un problème a eu lieu dans l'algorithme de mariage (Algo. 1 et 2), après au pire cas $2((n-1)^2+1)$ unités de temps, un homme va avoir parcouru sa liste de préférences entièrement et atteindre la fin. Dans ce cas, l'homme va enclencher un reset. Comme il y a autant d'homme que de femme, si aucune femme n'accepte l'appariement avec un homme à cause de la corruption, il arrive au début : à chaque refus (Ligne 16, il demande à la femme précédente (Ligne 2) jusqu'à la fin de la liste (Ligne 5).

Une fois qu'un homme détecte qu'il faut faire un reset, il utilise le module du reset qui le permet via la variable *invoke* (Algo 1, ligne 5). Grâce à celle-ci, le reset est annoncé et enclenché. Les variables *invoke* et *reset* sont donc à *true*, ce qui bloque le reste de l'algorithme.

Si plusieurs hommes déclenchent un reset simultanément, les demandes vont remonter à la racine. Cependant, il va y avoir un seul passage de reset, celui-ci étant auto-stabilisant, il gère les multiples demandes. Aprés l'annonce du reset, plus aucun homme ne peut utiliser l'algorithme de mariage et donc lancer de

reset. On sait aussi, que, comme le reset réinitialise les variables et vide les canaux et qu'il n'y a pas eu de réémission pendant le reset (prédicat), on repart d'une configuration adaptée au bon déroulement de l'algorithme de mariage.

Un reset va mettre les variables dans une configuration légale : mise à n de $pref\_courante$ et $mariage\_actuel$ mis à $\varnothing$. Comme les variables de *reset* et *invoke* sont remises à false, l'algorithme de mariage peut reprendre : les hommes vont envoyer à leur femme préférée une `Proposition`. À chaque refus, ils vont demander à la précédente. Comme aucune erreur n'est survenue, une fois que les appariements sont faits, les hommes vont boucler sur les derniéres femmes pour vérifier l'absence de paires bloquantes. Aucune ne va être trouvée, l'homme ayant déjà demandé aux femmes mieux classées, il ne peut modifier son mariage. De leur cÃt'té, une fois mariée, une femme ne peut plus être seule, elle ne peux qu'améliorer son mariage. Ajouté au fait qu'il y a autant de nœuds dans chaque sous-ensemble, un homme ne peut atteindre à nouveau le début de la liste. (Puisque forcément il y aura une femme de disponible) Il ne peut donc y avoir à nouveau de reset après qu'un soit fini.

Complexité en messages La détection d'une erreur nécessite au plus $2(n-1)n+2(n-1) = 2n^2-2$ messages. En effet, il faut au pire que $n-1$ hommes parcourent leur listes (et donc que les femmes répondent, d'où le 2) jusqu'au bout et que le dernier envoie seulement $n-1$ messages (il n'envoie pas de message à la femme corrompue, auquel cas, le problème serait détecté avant) En ce qui concerne le reset, la remontée du problème se fait avec un seul message jusqu'à la racine. (Ce ne sont que les hommes qui détectent le problème).

Il faut ensuite $3*(n+n-1)$ messages pour la diffusion et la fin du reset. (n messages pour la diffusion aux hommes, $n-1$ pour la diffusion au femmes et cela trois fois)

Enfin, l'algorithme d'acceptation différée utilise maximum $2n(n-1)+2 = 2n^2-2n-2$ messages.

Soit au total un total de $4n^2+4n-4$ messages, donc une complexité en $O(n^2)$

Complexité en temps Le temps pour délivrer un message peut être arbitraire mais pour l'analyse de complexité, nous posons que chaque message met au plus 1 unité de temps pour arriver.

La détection d'une erreur nécessite au plus $(n-1)^2$ unité de temps.

Le reset nécessite 1 unité de temps pour l'annonce du reset puis 2 * 3 unité de temps pour se diffuser et finir, soit au total 7 unité de temps.

Le mariage se construit en $2n(n-1)+2 = 2n^2-2n-2$ messages en cascade donc autant en unité de temps.

Soit une complexité en $O(n^2)$

5 Conclusion

Nous avons construit un algorithme auto-stabilisant qui résout le problème du mariage stable dans un système à passage de message asynchrone. Celui-ci arrive au résultat avec une complexité de $O(n^2)$ unités de temps. Nous déclinons nos travaux sous 3 axes : l'amélioration du reset utilisé dans l'algorithme (*i.e.* sans recourir à la traduction automatique de modèle définie par Katz et Perry). D'autre part nous souhaitons proposer une solution alternative qui ne requiert pas de reset. Enfin nous souhaitons travailler sur une généralisation du problème de mariage stable : le problème des colocataires.

Références

[AB93] Y Afek and GM Brown. Self-stabilization over unreliable communication media. *Distributed Computing*, 7(1) :27–34, 1993.

[APSVD94] Baruch Awerbuch, Boaz Patt-Shamir, George Varghese, and Shlomi Dolev. *Distributed Algorithms : 8th International Workshop, WDAG '1994 Terschelling, The Netherlands, September 29 – October 1, 1994 Proceedings*, chapter Self-stabilization by local checking and global reset, pages 326–339. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[BM05a] Ismel Brito and Pedro Meseguer. Distributed stable marriage problem. In *6th Workshop on Distributed Constraint Reasoning at IJCAI*, volume 5, pages 135–147, 2005.

[BM05b] Ismel Brito and Pedro Meseguer. *Principles and Practice of Constraint Programming - CP 2005 : 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005. Proceedings*, chapter Distributed Stable Matching Problems, pages 152–166. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[Dij74] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11) :643–644, November 1974.

[DIM93] Shlomi Dolev, Amos Israeli, and Shlomo Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distrib. Comput.*, 7(1) :3–16, November 1993.

[Dol00] Shlomi Dolev. *Self-stabilization*. MIT Press, Cambridge, MA, USA, 2000.

[GS62] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1) :9–15, Jan. 1962.

[KP90] Shmuel Katz and Kenneth Perry. Self-stabilizing extensions for message-passing systems. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, PODC '90, pages 91–101, New York, NY, USA, 1990. ACM.

[KPS09] A. Kipnis and B. Patt-Shamir. A note on distributed stable matching. In *Distributed Computing Systems, 2009. ICDCS '09. 29th IEEE International Conference on*, pages 466–473, June 2009.

[Mat08] Fabien Mathieu. Self-stabilization in preference-based systems. *Peer-to-Peer Networking and Applications*, 1(2) :104–121, 2008.

[OR15] Rafail Ostrovsky and Will Rosenbaum. Fast distributed almost stable matchings. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 101–108, New York, NY, USA, 2015. ACM.

[Var94] George Varghese. Self-stabilization by counter flushing. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '94, pages 244–253, New York, NY, USA, 1994. ACM.

[XL11a] H. Xu and B. Li. Egalitarian stable matching for VM migration in cloud computing. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 631–636, April 2011.

[XL11b] H. Xu and B. Li. Seen as stable marriages. In *INFOCOM, 2011 Proceedings IEEE*, pages 586–590, April 2011.