Appariement stable décentralisé dans les communautés mixtes

Rapport de Master 2 Recherche
Web Intelligence

à l'Université Jean Monnet et l'École des Mines de Saint-Étienne

14 juin 2012

Lucas CERQUEIRA-MARTINS

Laboratoire d'accueil : École Nationale Supérieure des Mines de Saint-Étienne,
Institut Henri Fayol
Responsable de stage : Gauthier Picard
Equipe d'accueil : Département ISCOD

Mots-clés : Systèmes Multi-Agents, Marriage Stable, Équité, Privacité

Résumé : Ce rapport présente le problème classique des mariages stables, ainsi que les différentes approches nous permettant de résoudre ce problème. Pour chacun des différents algorithmes que nous abordons, nous porterons une grande attention sur plusieurs points : nous vérifions tout d'abord si ces solutions ne favorisent pas une communauté au détriment de l'autre, puis si les individus sont parvenus à conserver privées leurs informations personnelles, et enfin si une solution parvient dans tous les cas à être obtenue. Nous proposons également des outils se basant sur le bien-être social des individus afin de pouvoir comparer les différents algorithmes et évaluer les points forts de chacun d'eux. Suite aux différents résultats obtenus, nous nous intéressons plus particulièrement à l'algorithme de Casanova et proposons des modifications qui lui permet d'obtenir des résultats pour des situations posant actuellement problème.

Table des matières

Introduction

Le problème des mariages stables, ou *Stable Marriage Problem* (SM) en anglais, a été étudié pour la première fois par Gale et Shapley [8]. À partir de deux communautés composées d'un même nombre d'individus, l'objectif de ce problème est de former des couples entre les membres des deux communautés. Pour cela, nous disposons pour chaque individu d'une liste dans laquelle les membres de l'autre communauté seront triés par ordres de préférences. Ce problème a de nombreuses applications comme par exemple l'attribution de projet à des étudiants, où chaque étudiant classe les projets par préférences et réciproquement les responsables de chaque projet classent les candidats. Nous pouvons également citer comme exemple de domaines d'applications le covoiturage, la location d'appartements pour les vacances ou les réseaux d'entreprises.
L'objectif que nous nous sommes fixé est de produire des algorithmes assurant la privacité et l'équité. En effet, lors de l'application de ce problème dans des situations concrètes, il est souvent nécessaire que la solution atteinte ne favorise aucune des deux communautés afin qu'aucune d'elles ne trouvent le résultat injuste. De plus, il est également important parfois que les préférences d'un individu soit confidentielles, tout comme le choix final de son partenaire. Pour cela, nous analyserons différents algorithmes en accordant une attention toute particulière envers les algorithmes décentralisés qui permettent pour la plupart facilement d'atteindre cet état de privacité au cours de son exécution.
Pour cela, nous avons tout d'abord analysé les différents algorithmes proposés permettant de résoudre ce problème. Par la suite, nous nous sommes intéressés aux différentes propriétés que garantissaient ces algorithmes, telles que la privacité, ou encore l'équité des solutions obtenues, afin de pouvoir comparer entre eux ces différents algorithmes. Pour cela, nous avons implémenté certains de ces algorithmes afin de pouvoir en extraire des statistiques que nous pourrions comparer à celles obtenues grâce aux algorithmes déjà implémentés. Pour finir, nous nous sommes intéressés aux différentes modifications que nous pourrions apporter à certain de ces algorithmes afin d'obtenir de nouvelles propriétés ou d'améliorer les résultats obtenus.

1 Problème du mariage stable

1.1 Définition d'un problème de mariage stable

Nous étudions ici le problème des mariages stables, ou *Stable Marriage Problem* en anglais, qui a été présenté par Gale et Shapley[8]. Une instance d'un problème de mariage stable de taille n consiste en deux communautés de n individus que nous appellerons les hommes et les femmes. Chaque individu aura à sa disposition une liste de préférences dans laquelle sera classée chaque membre de l'autre communauté suivant leur préférence. On considère que l'ordre dans ces listes de préférences est strict, c'est à dire que chaque individu est capable de faire un choix entre deux partenaires (il n'y a ni égalité, ni indifférence dans ces listes de préférence). Dans une instance classique du problème de mariage stable, les listes de préférences sont complètes (il ne manque aucun individu de l'autre communauté dans la liste). Il existe en revanche une version du problème de mariage stable appelé problème de mariage stable avec liste incomplète (SMI) dans lequel au moins une personne possède une liste de préférence incomplète. Cela signifie que cette personne préfère être seule plutôt qu'avec un des individus non-listés.

$$\begin{aligned} m_1 &: w_3 w_1 w_2 & \qquad w_1 &: m_1 m_2 m_3 \\ m_2 &: w_1 w_3 w_2 & \qquad w_2 &: m_2 m_3 m_1 \\ m_3 &: w_3 w_2 w_1 & \qquad w_3 &: m_1 m_2 m_3 \end{aligned}$$

Exemple 1. Exemple d'instance d'un problème SM de taille 3. Les liste de préférence sont triés dans l'ordre décroissant, les partenaires préférés sont les premier de la liste.

On appelle appariement, ou en anglais *matching*, l'ensemble de n mariages obtenu en mariant chaque homme avec une et une seul femme, réciproquement, chaque femme sera mariée avec un et un seul homme. L'appariement est considéré comme stable uniquement s'il ne contient aucun couple bloquante, comme défini ci-après :

Définition 1. *Soit un problème SM ayant pour communauté X et Y, soit M un appariement pour ce SM, et soient m_i un membre de la communauté X et w_i un membre de la communauté Y. Un couple (m_i,w_i) est bloquant dans l'appariement M si :*

- *m_i et w_i ne sont pas mariés dans M ;*
- *m_i préfère w_i à sa partenaire actuelle dans M ;*
- *w_i préfère m_i à son partenaire actuel dans M.*

Résoudre un problème SM revient alors à obtenir un appariement stable. Pour les problèmes SMI, tous les individus ne peuvent pas forcement trouver un partenaire, donc un appariement peut être une solution au problème sans être complet, en revanche, pour chaque individu, leur partenaire doit faire partie de leur liste de préférence.

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

Exemple 2. Exemples d'appariements pour l'instance présentée dans l'exemple 1. L'appariement de gauche est stable, en revanche, la paire (m_2, w_1) est bloquante dans l'appariement de droite, qui n'est donc pas stable.

1.2 Algorithme de Gale-Shapley

L'algorithme de Gale-Shapley est une preuve constructive que pour chaque instance de SM ou SMI, il existe au moins un appariement stable qui soit solution de ce problème.
Cet algorithme peut se décrire comme une cérémonie au cours de laquelle les femmes attendent dans une salle et les hommes sont à l'extérieur. Dans un ordre aléatoire, chaque homme entre dans la salle à tour de rôle et fait une proposition à la femme qu'il préfère. Si cette femme est libre et le considère acceptable (i.e. il est dans sa liste de préférences), il devient alors son partenaire. Si cette femme possède déjà un partenaire, elle va choisir la personne qu'elle préfère entre son partenaire actuel et le proposant. La personne choisit reste alors que la personne rejetée sort de la pièce. Cette personne reviendra plus tard faire une nouvelle proposition à la femme suivante dans sa liste de préférences. On s'arrête lorsque tous les hommes sont mariés ou quand les hommes restants sont désespérés (il n'ont plus de partenaire potentiel dans leur liste de préférences).
Dans cette version de l'algorithme GS, les hommes proposent et les femmes disposent. Étant donné que les hommes sont les seules personne à faire des propositions, on dit que GS est orienté homme. On peut aisément obtenir une version orientée femme de cet algorithme en inversant le rôle des hommes et des femmes dans cet algorithme. Dans ce cas, ce sont les femmes qui proposent et les hommes qui disposent.
Lors du déroulement de la version orienté homme de l'algorithme de GS, il a été démontré que les solutions obtenues sont les plus avantageuses pour les hommes parmi toutes les solutions possibles. Inversement, ces solutions obtenues sont les moins avantageuses pour les femmes puisque contrairement aux hommes qui font leur choix parmi les premiers individus de leurs listes de préférences, les femmes elle se contentent de choisir parmi les propositions reçues.

Data : SM
Result : M
Tous les individus sont libres;
while *il existe un homme m qui ne soit pas désespéré* **do**
  $w \leftarrow$ première femme de la liste de m à qui il ne s'est pas proposé;
  // m propose à w;
  $m_2 \leftarrow$ le partenaire actuel de w éventuellement $null$;
  if *w considère m comme acceptable* **then**
    Marier m et w;
    //w dispose;
    if m_2 *existe (i.e. n'est pas null)* **then**
      $m_2 \leftarrow$ libre;

Algorithme 1 : GS orienté homme

1.3 Problèmes similaires et extensions

Il existe de nombreux problèmes dont le principe est proche de celui d'un problème SM comme par exemple le problème de SMI. Nous ne nous intéresserons pas ici aux algorithmes nous permettant de résoudre ces différent problème, mais de part la nature semblable de ces problèmes, ces algorithmes sont très proche de ceux pouvant résoudre les problèmes SM.

Le *problème des admissions d'universités* a été présenté pour la première fois par Gale et Shapley dans [Lien]. Le problème SM a même été présenté ensuite comme un cas particulier ou une simplification de ce problème. Le problème ce présente de la façon suivante. On dispose d'un certain nombre d'universités qui doivent s'occuper des admissions de nouveaux étudiants. On dispose également d'un certain nombre d'étudiants souhaitant rejoindre une université et possédant une liste de préférences dans laquelle chaque étudiant va trier les universités dans lesquelles il souhaiterait être admis de son premier choix jusqu'à son dernier choix. Chaque université a un quota précis d'étudiants qu'elle peut recevoir, et dispose également d'une liste de préférence dans laquelle seront triés les étudiants souhaitant rejoindre cette université. L'objectif sera alors, en supposant qu'il y ait suffisamment de place disponible dans les universités, d'attribuer une université à chaque étudiant, sachant qu'un étudiant pourrait annuler sa candidature à une université s'il reçoit une acceptation dans une autre université qu'il préfère. Dans un même temps des places pourraient se libérer dans des universités suite à ces annulations, et certains étudiants auparavant rejetés pourraient devenir des candidats acceptables.
On remarque alors que dans un cas particulier ou chaque université ne pourrais accepter qu'un seul étudiant et ou le nombre d'universités serait identique au nombre d'étudiants, nous nous retrouverions alors dans la situation d'un problème SM.

Le *problème des colocataire* est pour sa part une variante du problème SM dans laquelle on ne disposerait que d'une seule communauté. Chaque membre de la communauté cherche un partenaire afin de devenir son colocataire, et dispose pour cela d'une liste de préférences dans laquelle chaque membre de sa communauté excepté lui-même est trié par préférence. La solution que l'on cherche à obtenir est un appariement qui associerait chaque individu de cette communauté avec un unique partenaire. On notera alors que même si le problème est semblable à un problème SM, une approche tel que l'algorithme GS est impossible puisque nous ne pouvons séparer cette unique communauté en deux sous-communauté, une de proposants et une autre de disposants, sans que des problèmes de couples bloquants apparaissent entre membres de même sous-communautés.

Le *problème SM avec égalité* dans les listes de préférences entraîne des variations dans la définition de couple bloquant et par extension la définition de stabilité. Il existe dans ce cas trois niveaux de stabilité différents :

- **Stabilité faible** : dans ce niveau de stabilité, la définition de couple bloquant reste identique à celle utilisée dans un problème SM. On peut donc obtenir une solution faiblement stable en choisissant arbitrairement un ordre entre les individus classés au même niveau dans une liste de préférence et en utilisant un algorithme pouvant résoudre un problème SM classique tel que l'algorithme GS.
- **Stabilité forte** : dans ce niveau de stabilité, la définition de couple bloquant est étendue de telle sorte qu'un couple (m,w) est bloquant dans M s'il ne sont pas mariés dans M et si l'un préfère strictement l'autre à son partenaire actuel, tandis que l'autre préfère le premier à son partenaire actuel ou est indifférent entre eux deux (les deux sont à égalité dans la liste de préférence).

– **Super stabilité** : dans ce niveau de stabilité, la définition de couple bloquant est étendue de telle sorte qu'un couple (m,w) est bloquant dans M s'il ne sont pas mariés dans M et si chacun d'eux préfère l'autre à son partenaire actuel ou est indifférent entre eux deux (les deux sont à égalité dans la liste de préférence).

Dans le cas d'un problème SM avec égalité, il faut savoir que même s'il existe toujours au moins une solution faiblement stable, il n'existe en revanche pas toujours d'appariement fortement stable ou super stable.

2 Algorithmes de l'état de l'art

Dans ce chapitre, nous allons nous intéresser à cinq algorithmes, (six si l'on ajoute GS qui a été présenté plus tôt), qui couvrent l'ensemble des solutions proposées afin de résoudre un problème SM.
Pour commencer, nous étudierons les algorithmes centralisés, à savoir SML et Zig-Zag. Le premier repose sur une recherche locale de solution par minimisation du nombre de couples bloquants, le second repose sur la création d'une matrice dans laquelle tous les couples possibles seront stockés, puis consistera à parcourir cette matrice afin de choisir quels couples marier.
Les trois algorithmes décentralisés que nous étudierons sont DisEGS, DisFC et Casanova. L'algorithme DisEGS est une version distribuée de GS avec quelques modifications mineures permettant aux individus de modifier leur liste de préférence durant l'exécution de l'algorithme. DisFC pour sa part, propose de transformer une instance SM en CSP et d'utiliser leur algorithme afin de résoudre ce CSP en maintenant la privacité. Casanova pour sa part propose une approche différente où tous les agents quelle que soit leur communauté font des propositions et traitent celles qu'ils reçoivent, en utilisant une stratégie de concession minimale afin d'obtenir le meilleur partenaire possible.
En dehors du critère de décentralisation, chacun de ces algorithmes sera analysé suivant trois autres critères : la privacité, l'équité et la complétude. La privacité représente le fait que chaque agent ne connaisse que son propre partenaire et n'a aucune information sur les appariements qui ont été formés par les individus restant. L'équité représente le fait qu'un algorithme ne porte aucune attention lors de son exécution à la communauté à laquelle appartient chaque individu. Ainsi, chaque individu sera traité de la même façon quelque soit sa communauté, ce qui se reflétera le plus souvent par un appariement dans lequel la satisfaction des individus de chaque communauté sera proche. La complétude nous sert à garantir que lorsqu'un algorithme est exécuté, il parviendra forcément à trouver un appariement stable comme solution. Un algorithme ne sera pas complet quand il renverra parfois un appariement non stable, ou ne parviendra pas à obtenir, à la fin de son exécution, un appariement dans lequel tous les individus sont mariés.

2.1 SML

L'algorithme SML[9] (pour SM Local search) utilise la recherche locale de solutions pour résoudre un problème SM. Pour cela, à partir d'un instance d'un problème SM, on construit un appariement aléatoirement. Une fois ce premier appariement obtenu, on va effectuer un certain nombre de pas de recherche afin de minimiser le nombre de couples bloquants existant dans l'appariement jusqu'à le rendre stable. En revanche, si l'appariement n'a pas pu être rendu stable après un nombre de pas qui a été fixé à l'origine, l'algorithme s'arrête alors sans avoir obtenu de solution, ce qui rend cet algorithme non complet.

À chaque pas de recherche le travail qui est effectué consiste à rechercher la liste de tous les couples bloquants présents dans l'appariement. Une heuristique est utilisée pour choisir un couple parmi tous les couples bloquants, puis marie ensemble les deux individus qui composent le couple bloquant, ensuite, les deux anciens partenaires des individus qui composent le couple bloquant sont à leur tour mariés ensemble. Pour finir, on vérifie si le nouvel appariement obtenu est stable, dans le cas contraire, un nouveau pas de recherche est effectué.
L'heuristique utilisée pour choisir le couple bloquant à marier s'effectue en deux temps. Dans un premier temps, l'algorithme va effectuer une vérification sur la liste des couples bloquants, et va rechercher les couples bloquants qui aurait en commun un individu. Lorsque l'on trouve plusieurs couples bloquants ayant en commun un individu, cela revient à dire que cet individu n'est pas satisfait avec son partenaire actuel et a plusieurs candidats potentiels avec qui il est prêt à se marier. L'algorithme va alors chercher parmi la liste de préférences de cet individu son partenaire potentiel préféré, et va alors conserver dans la liste des couples bloquants potentiel le couple bloquant composé de cet individu et de son partenaire préféré, les couples que cet individu formait avec ses autres partenaires potentiel sont pour leur part supprimés de la liste. L'argument qui est utilisé pour justifier ce choix est que si un de ces couples avait été choisit dans le pas de recherche et avait été marié ensemble, le couple bloquant formé par cet individu et son partenaire préféré serait toujours présent et finirait par être choisi à son tour. Afin de minimiser le nombre de pas de recherche effectué, on préférera marier directement cet individu avec son partenaire potentiel préféré.
Dans un deuxième temps, l'heuristique sélectionnant le couple bloquant à marier va récupérer la nouvelle liste de paires bloquantes obtenus après l'étape précédente, et va simplement sélectionner le couple bloquant qui générerait le moins de couples bloquants au pas de recherche suivant si ses membres étaient mariés ensemble. Si l'algorithme se contentait à cette étape de marier à chaque fois le couple bloquant minimisant le nombre de couples bloquants au pas suivant, on risquerait de bloquer autour d'un minimum local sans explorer de nouvelle solution. Pour éviter cela, en respectant une probabilité p, on va sélectionner aléatoirement le couple bloquant à marier au lieu de celui qui aurait été choisit normalement par l'heuristique.

Cet algorithme n'est pas complet puisqu'il n'est pas assuré d'obtenir une solution stable, en revanche, puisque son approche ne favorise aucune des deux communautés, le résultat obtenu par cet algorithme sera équitable. Enfin, concernant la privacité, puisque l'algorithme est centralisé, cette condition n'est pas atteinte puisque la liste de tous les couples formés sera stocké au même endroit.

2.2 Zig-Zag

Zig-Zag[13] est le nom d'un ensemble d'algorithmes où chacun des algorithmes proposés est une amélioration du précédent. Le principe de base de tous ces algorithmes repose sur une matrice de mariages qui est une représentation d'un problème SM que les auteurs ont également proposée. Cette matrice est une matrice carrée de taille $n + 1$ ou n est la taille du problème SM. L'objectif est alors de placer tous les couples possibles dans la matrice de mariage, ce qui nous donne un total de n^2 couples à placer dans cette matrice. Pour savoir dans quel case placer un couple, on va s'intéresser à la position du conjoint dans la liste de préférences de chacun des deux individus. Ainsi, si nous cherchons à placer le couple (m_i, w_j), nous allons noter k la position de w_j dans la liste de préférences de m_i, et inversement nous noterons l la position de m_i dans la liste de préférences de w_j. Le couple (m_i, w_j) sera alors placé à la case de coordonnée (k, l). Les lignes du tableau représentent donc la position des partenaires des hommes dans leurs listes de préférences, de 1 à n en commençant par le bas. la dernière ligne, noté

∞ contient les couples formés avec les partenaires n'apparaissant pas dans la liste de préférences d'un homme, ce qui n'arrive que dans le cas d'un problème SMI. De la même façon, les colonnes représentent donc la position des partenaires des femmes dans leurs listes de préférences. Nous noterons qu'une fois tous les couples possible placés dans la matrice, certaine cases pourraient contenir plusieurs couples, alors que d'autres pourraient être vides.

∞				
3	(m_2, w_2)		(m_3, w_1) (m_1, w_2)	
2	(m_1, w_1)	(m_3, w_2) (m_2, w_3)		
1	(m_1, w_3)	(m_2, w_1)	(m_3, w_3)	
	1	2	3	∞

Exemple 3. Exemple de matrice de mariages pour l'instance présentée dans l'exemple 1.

Pour la suite et pour plus de clarté, nous déciderons d'appeler diagonale la droite passant par l'angle supérieur gauche et par l'angle inférieur droit. Inversement nous appellerons anti-diagonale la droite passant par l'angle inférieur gauche et supérieur droit. L'objectif que les auteurs se sont fixé est d'obtenir des appariements équitables grâce à cet table. Pour cela, l'idéal serait alors de marier ensemble des couples dans la matrice placés près de l'anti-diagonale passant par la case (1,1), puisqu'en effet, tous les couples placés sur cette anti-diagonale ont leur deux membres sont au même niveau dans la liste de préférence de leur conjoint. Dans un même temps, si l'on cherche à obtenir un appariement dont la satisfaction générale des individus est élevée, il faut privilégier les couples placés dans le quart inférieur gauche de la matrice de mariages.

La première version d'algorithme se reposant sur la matrice de mariage qui a été proposé fut nommée Zig-Zag (ZZ). Dans cette version, nous allons parcourir toutes les cases de la matrice en se déplaçant le long des diagonales traversant cette matrice. On commence par la diagonale passant par la case $(1, 1)$, puis celle passant par la case $(2, 1)$, et s'éloignant ainsi de plus en plus de l'angle inférieur gauche de la matrice. Concernant le sens de parcours des diagonales, afin de ne pas favoriser une communauté plutôt qu'une autre en se déplaçant toujours dans le même sens, il a été décider d'alterner le sens de parcours après chaque diagonale, ce qui nous donne l'impression de zigzaguer sur la matrice de mariage, d'où le nom de l'algorithme. Pendant ce parcours de la matrice, l'appariement qui sera la solution au problème est construit en respectant une règle simple : à chaque fois qu'un couple est repéré pendant le parcours de la matrice, si les deux membres de ce couple sont actuellement célibataires, ils sont mariés l'un avec l'autre, dans le cas contraire, le couple est ignoré. L'algorithme s'arrête alors lorsque la matrice a été consulté en intégralité. Dès lors, tous les individus ont été mariés et un appariement a été construit.
Après avoir comparé les appariements obtenus par ZZ avec ceux obtenus par GS, il a été constaté que ZZ obtient de meilleurs résultats, que ce soit concernant la satisfaction générale ou l'équité, en revanche, le problème principal est qu'il n'y a aucune garantie que l'appariement obtenue soit stable, et donc que la solution soit valide. L'algorithme ZZ existe en deux versions, soit homme-optimal, soit femme-optimale suivant le sens de parcours qui a été choisi pour la première diagonale.

La version suivante qui a été proposé a pour but de modifier ZZ afin de ne plus avoir deux versions de ZZ mais une version unique de ZZ dont le sens de parcours initial n'ai plus d'incidence. Pour cela, les auteurs ont proposé l'algorithme Optimal Zig-Zag (OZ), dans lequel le parcours s'effectue toujours le long des diagonales et en parcourant les diagonales dans le même ordre. En revanche, plutôt que de parcourir chaque diagonale d'une extrémité à l'autre, l'idée est de commencer par le milieu de la diagonale puis de s'étendre vers ses extrémités. En dehors du parcours différent de la matrice, le reste de l'algorithme est identique à ZZ.
Concernant les résultat, OZ est légèrement moins performant en terme de satisfaction générale ou d'équité par rapport à ZZ, en revanche, il réduit de façon significative le nombre de couples bloquants restants.

Le dernier algorithme présenté, qui fut appelé Blocked Zig-Zag (BZ), est une extension de OZ. Pour commencer, BZ construit un appariement en utilisant l'algorithme OZ. Cependant, une fois cet appariement obtenu, BZ va parcourir à nouveau la matrice des mariages. Lors de ce nouveau parcours de la matrice, BZ va rechercher les couples qui sont actuellement bloquants. Si une paire bloquante est trouvée, ses deux membres sont mariés ensemble, ainsi que leurs deux anciens partenaires qui sont également mariés ensemble (de la même façon que dans l'algorithme SML), les couples ne formant pas un couple bloquant sont ignorés. Ce parcours à la recherche de couple bloquant est effectué plusieurs fois, jusqu'à atteindre un appariement stable ou atteindre un nombre maximum de parcours fixé au préalable. La satisfaction général et l'équité de BZ est du même niveau que pour OZ, en revanche, BZ arrive maintenant à atteindre un appariement stable dans environ 95% des cas. En revanche, alors que la complexité de ZZ et OZ est de l'ordre de $O(n^2)$, la complexité de BZ est pour sa part est de l'ordre de $O(n^3)$.
En conclusion, même si BZ obtient de bons scores en général et permet de satisfaire l'équité, il est actuellement impossible de garantir qu'une solution stable sera trouvée à l'aide de BZ, ce qui empêche d'atteindre l'état de complétude. Concernant la privacité, cette condition n'est pas satisfaite puisque les couples sont formés lors du parcours de la matrice de mariages, et sont donc connus de tous.

2.3 DisEGS

L'algorithme DisEGS[3] est une version distribuée de l'algorithme EGS pour *Extended Gale Shapley* qui est lui même une variante de l'algorithme GS. L'algorithme EGS à un déroulement identique à celui de GS en dehors du fait que pendant son déroulement les individus vont mettre à jour leur liste de préférence en suppriment de cette liste les partenaires avec lesquels ils n'ont aucune chance de se marier. Pour les hommes, cela se produira lorsqu'ils seront rejetés par un partenaire potentiel, ils n'auront alors qu'à supprimer ce partenaire de leur liste de préférence. En mettant ainsi à jour leur liste de préférences, les hommes feront toujours leur proposition à la première personne de leur liste de préférence lorsqu'il auront une proposition à faire.
Pour les femmes, cela se produira à chaque fois qu'elle recevra une proposition acceptable. Puisqu'une fois en couple, une femme n'acceptera que les propositions meilleures que celle de leur partenaire actuel, elle peut donc à chaque proposition acceptable reçue, retirer de sa liste de préférences les personnes moins bien classées que leur partenaire actuel. Par extension, elle peut également prévenir les personnes qu'elle vient ainsi de supprimé qu'il leurs est inutile de lui faire une proposition. Ainsi, dans cette situation, ces hommes supprimeront de leurs listes des personnes à qui ils n'ont pas encore fait de proposition, mais qui n'accepterait pas de toute façon cette proposition.

L'algorithme DisEGS (cf Algorithme 2), est une version multi-agent de l'algorithme EGS, dans lequel chaque membre d'une communauté est représenté par un unique agent. Chaque agent membre d'une même communauté aura le même algorithme, en revanche, il ne sait rien de l'état des agents de l'autre communauté. Chaque agent communiquera avec les membres de l'autre communauté en lui envoyant des messages respectant un protocole de communication qui a été défini au préalable dans l'algorithme. En revanche, deux membres d'une même communauté ne vont jamais communiquer l'un avec l'autre. Tout comme l'algorithme GS, la version présentée ici est orientée homme, et on obtient une version orientée femme en inversant le fonctionnement des agents dans les deux communautés. De la même façon, la version orientée homme donne les résultats favorisant le mieux possible les hommes et les moins bien possible pour les femmes.
Il existe un agent spécial qui ne représente aucun individu, dont le seul rôle sera de surveiller le déroulement de l'algorithme dans son ensemble et d'envoyer à chaque agent un message lorsqu'une solution à été trouvée, prévenant ces agents qu'il peuvent arrêter leurs recherches. Le partenaire actuel de chaque agent sera son partenaire final.

Grâce à cette approche distribuée, cet algorithme obtient une propriété très intéressante pour ce problème qui est la privacité des résultats. En effet, chaque agent au final ne connaît que son partenaire, et ne connaît aucun autre mariage présent dans l'appariement obtenue. En revanche, puisque les hommes proposent et les femmes disposent, cet algorithme n'est pas équitable puisqu'il favorise les hommes. Pour finir, cet algorithme parvient dans tout les cas à atteindre un appariement stable, l'état de complétude est atteint.

2.4 DisFC

L'algorithme DisFC[3], pour *Distributed Forward Checking*, propose une approche différente dont l'objectif est d'utiliser une solution distribuée afin de maintenir la privacité des résultats. Pour cela, leur algorithme consiste dans un premier temps à reformuler à un problème SM pour le transformer en un problème de satisfaction de contrainte, et consiste ensuite à résoudre ce CSP en utilisant l'algorithme de *Forward Checking*[2].

DisFC est un algorithme multi-agent qui a été créé dans le but de résoudre les CSP en conservant secrète la valeur choisit par chaque agent par rapport aux autres agents. Lors du déroulement d'un algorithme distribué pouvant résoudre un CSP, telle que par exemple l'algorithme de *Asynchronous Backtracking*[12], une hiérarchie est établie entre chaque agent de sorte que les agents choisissent leurs valeurs en commençant par les agents de priorité la plus élevé. Les agent de priorité inférieur devront donc pour leur part prendre leur décision de façon à ne pas contredire un choix qui aurait effectué par un agent de priorité supérieur autant que possible. La principale différence entre DisFC et les autres algorithmes résolvant les CSP est qu'une fois qu'un agent à choisit une valeur à prendre, au lieu d'envoyer cette valeur aux agents de priorité inférieure afin qu'elles puissent à leur tour faire leur choix, l'agent calcule pour chacun de ces agents de priorité inférieure les valeurs qu'ils peuvent prendre et leurs envoie une liste de valeurs possible, ainsi qu'une variable dont la valeur correspond au nombre de choix que l'agent a effectué jusqu'à présent. Cette valeur, que l'on initialise à 0, est incrémenté à chaque fois que l'agent décide de prendre une nouvelle valeur. De cette façon, lorsque cette agent recevra un message d'un agent de priorité inférieur, il pourra vérifier si cet agent à bien mis à jour ses informations en se basant sur les choix de valeurs les plus récents. Chaque agent choisit alors sa valeur à prendre parmi celles qui lui sont autorisées par les agents de priorité supérieure, sans jamais savoir exactement la valeur qui a été prise par ces agents. Si aucune valeur

```
procedure Man( )
m ← free;
end ← false;
while ¬end do
    if m = free and list(m) ≠ ∅ then
        w ← first(list(m));
        sendMsg(propose,m,w);
        m ← w;
    msg ← getMsg();
    switch msg.type do
        case accept
            do nothing;
        case delete
            list(m) ← list(m) − msg.sender;
            if msg.sender = w then m ← free
        case stop
            end ← true;
```

```
procedure Woman( )
w ← free;
end ← false;
while ¬end do
    msg ← getMsg();
    switch msg.type do
        case propose
            m ← msg.sender;
            if m ∉ list(w) then
                sendMsg(delete,w,m);
            else
                sendMsg(accept,w,m);
                w ← m;
                foreach p after m in list(w) do
                    sendMsg(delete,w,p);
                    list(w) ← list(w) − p;
        case stop
            end ← true;
```

Algorithme 2 : DisEGS orienté homme

n'est disponible pour un agent, il va alors contacter ses agents de priorité supérieure pour leur signaler qu'il est impossible de résoudre le CSP avec les valeurs actuelles et qu'il leurs faut qu'ils choisissent une autre valeur. De cette façon, les agents parviennent à résoudre le CSP tout en conservant privé leur propre valeur.

Le problème SM peut être modélisé sous la forme d'un CSP binaire, c'est à dire un CSP ou chaque contraintes ne fait intervenir que deux variables à la fois. Dans cette modélisation, chaque personne est représenté par une variable et le domaine des valeurs qu'il peut prendre est représenté par sa liste de préférences. Tout agent partage alors une et une seule contrainte avec chaque membre de la communauté opposée, et ces contraintes seront les seules présentes dans ce CSP. Chaque contrainte C_{ij} qui implique deux individus m_i et w_j nécessiterait de connaître les listes de préférences de chacun de ces deux individus pour être construite en intégralité. Or, comme chaque individu ne connaît que sa propre liste de préférences, les individus ne connaîtront qu'une version partielle de leur contrainte. Ainsi, on note $C_{i(j)}$ la contrainte telle qu'elle est connue par l'agent m_i et $C_{(i)j}$ celle connu par l'agent w_i.
Dans DisFC, la hiérarchie est construite de sorte que les hommes sont toujours des agents de priorités supérieures par rapport aux femmes. Ainsi, on obtient un déroulement de l'algorithme proche de celui de GS, à savoir que les hommes choisissent leurs partenaires puis envoient pour chaque femmes la liste des valeurs que les femmes sont autorisés à prendre suite à ce choix. Les femmes pour leur part, se contentent de choisir un partenaire en fonction des décisions prisent par les hommes et envoient un message à un homme lorsque leur choix est impossible suite aux choix effectués par les autres hommes.
Chaque contrainte est formulée sous la forme d'une matrice de taille n où chaque ligne représente une femme et chaque colonne représente un homme trié par indice croissant. Chaque case (k, l) d'une matrice pourra avoir trois valeur différente : un 0 indiquera que le couple (m_l, w_k) ne pourra pas se marier ensemble alors qu'une case ayant pour valeur 1 indiquera que ce couple pourra se marier. Le dernier choix possible pour le contenue d'une case est le caractère ? qui signifie que la contrainte vue par un seul agent n'est pas suffisante pour prendre une décision. Il faudra alors combiner le contenu de la case dans la contrainte $C_{i(j)}$ et $C_{(i)j}$ pour obtenir un résultat : si le ? d'une version de la contrainte est combiné avec un ? ou un 0 de l'autre version de la contrainte, on considère que la case à pour valeur 0. En revanche, si un ? est combiné avec un 1, on considère le contenu de la case comme étant un 1. Ainsi, lorsqu'un homme choisit une partenaire, pour envoyer la liste des choix que peut effectuer une femme, il lui enverra une ligne précise de sa matrice. En comparant cette ligne reçu avec celle contenue dans sa propre version de la contrainte, une femme aura une ligne remplie uniquement de 0 et de 1 qui lui permettra de savoir si elle est autorisée ou non à se marier avec l'homme associé à la colonne dont une case de la ligne est issue.
Concernant le remplissage des matrices, pour une contrainte $C_{i(j)}$ ou $C_{(i)j}$, toute les cases sur la ligne associé à w_j ainsi que les cases sur la colonne associé à m_i ont pour valeur 0 à l'exception de la case de (j, i) qui contiendra un 1. Cela signifiera que si m_i et w_j décidaient de se marier ensemble, ils ne pourraient pas se marier avec quelqu'un d'autre en même temps. Pour les cases restantes, le remplissage s'effectue différemment suivant si l'agent représente un homme ou une femme. Si l'agent représente l'homme m_i et cherche à remplir la contrainte $C_{i(j)}$ qu'il partage avec la femme w_j, alors une ligne k représentant la femme w_k sera remplie de 1 si m_i préfère la femme w_k à la femme w_j d'après sa liste de préférences, dans le cas contraire, la ligne sera rempli de ?. En revanche, dans les deux cas, les cases déjà remplies avec des 0 dans l'étape précédente restent inchangées. Si l'agent représente la femme w_j et cherche à remplir la contrainte $C_{(i)j}$ qu'il partage avec l'homme m_i, alors une colonne k représentant l'homme m_k sera remplie de 1 si w_j préfère l'homme m_k à l'homme m_i d'après sa liste de préférence, dans le cas contraire, la ligne sera remplie de ?. À nouveau, dans les deux cas, les cases déjà remplies avec des 0 restent in-

changées.

$C_{2(2)}$	m_1	m_2	m_3
w_1	1	0	1
w_2	0	1	0
w_3	?	0	?

$C_{(2)2}$	m_1	m_2	m_3
w_1	1	0	?
w_2	0	1	0
w_3	1	0	?

Exemple 4. Exemple de contrainte liant m_2 et w_2 de l'instance présentée dans l'exemple 1. Chacun de ces tableau montre la contrainte tel qu'elle est perçue par un des deux agents

Le déroulement de l'algorithme s'effectue donc de la façon suivante. Tout d'abord, chaque agent va construire sa version de chacune des contraintes qu'il partage avec les agents de l'autre communauté. Ensuite, chaque homme va choisir une partenaire en privilégiant la mieux placée dans sa liste de préférence. Après cela, chaque homme m_i qui aura choisi de se marier avec la femme w_k va envoyer à toutes les femmes w_j la ligne correspondant à la femme w_k dans sa contrainte $C_{i(j)}$. Lorsqu'une femme w_j reçoit une ligne de la part d'un homme m_i, elle va tout d'abord autoriser à nouveau les choix qui lui auraient été interdits par l'ancienne valeur de m_i, elle va ensuite comparer la ligne reçue avec celle existante dans sa contrainte $C_{(i)j}$ pour transformer les éventuels ? qui pourrait être présents dans la ligne en 0 ou 1. Pour finir, elle va analyser le contenu de la ligne pour savoir quelles valeurs lui sont interdites. Comme dans la matrice les colonnes sont triées par indice des hommes croissant, la première valeur de la ligne reçue indique si elle peut se marier avec m_1, la deuxième si elle peut avec m_2 et ainsi de suite jusqu'à la dernière valeur pour m_n. Si aucun choix ne lui est autorisé (par exemple, quand deux hommes différents veulent se marier avec elle, lui interdisant ici à chaque fois de se marier avec un autre homme) elle va alors indiquer aux hommes que leurs choix actuels ne convient pas et simplement attendre leurs nouveaux choix et la ligne qu'elle recevra en même temps pour chercher une valeur valide.
On s'intéresse maintenant aux agents m_2 et w_2 ainsi qu'à leurs contraintes telles que définies dans l'exemple 4. Si m_2 décidait de se marier avec w_3, il enverrait une ligne de la forme (?,0, ?) à w_2. w_2 pour sa part parviendrait à transformer cette ligne en (1,0,0) après comparaison avec sa propre vision de sa contrainte. w_2 en déduit donc qu'elle peut se marier avec m_1, mais pas avec les autres hommes.

Puisque dans l'algorithme DisFC les hommes choisissent et les femmes se contente de mettre les hommes d'accord entre eux, le résultat obtenue favorise grandement les hommes au détriment des femmes comme les algorithmes tels que GS le font également, ce qui l'empêche d'être équitable. En revanche, l'objectif que les auteurs s'étaient fixé, à savoir maintenir la privacité des choix, est bien atteint puisqu'un agent ne connaît que le choix qui a été effectué par l'individu à qui il est associé. En revanche, le nombre de messages envoyés est beaucoup plus important que celui des autre algorithme distribué permettant de résoudre un problème SM tel que DisEGS par exemple, ce qui rend la recherche d'une solution avec DisFC très lente, ce qui rend son utilisation plutôt compliquée dès lors que l'on commence à gérer plus d'une dizaine d'individus. Enfin, puisque l'algorithme va parcourir tous les appariements possibles, il va obligatoirement trouver au moins un appariement stable, cet algorithme atteint donc l'état de complétude.

2.5 Casanova

L'algorithme Casanova[7] est un algorithme multi-agent dans lequel chaque individu, quelle que soit la communauté à laquelle il appartient, aura le même comportement. Ainsi, contrairement à d'autre algorithmes tel

que GS où le comportement des hommes et des femmes est distinct, avec Casanova, chaque individu, qui sera représenté par un agent, fera des propositions aux autres agents et répondra à celles qu'il a reçu en même temps. Pour cela, chaque agent va suivre une stratégie de concession minimale basée sur son regret, afin de converger vers un appariement qui soit équitable pour les deux communautés. Ainsi, chaque agent commencera avec un niveau de regret de 1, et enverra une proposition à son partenaire préféré dans la liste de préférence. Si ce partenaire refuse, l'agent incrémente son regret de un niveau et enverra une proposition à ses deux partenaires préférés, et ainsi de suite.

Pendant l'exécution de l'algorithme, chaque agent pourra passer par plusieurs états différents :

- **free** est l'état initial d'un agent. Il correspond à un agent qui est actuellement en train de faire ses propositions, mais qui n'a de son côté reçu aucune proposition intéressante.
- **hesitating** est l'état d'un agent qui a reçu une proposition intéressante, mais qui n'a pas encore reçu toutes les réponses aux propositions qu'il a envoyées.
- **engaged** est l'état d'un agent qui a reçu une proposition intéressante, qui a reçu des réponses négatives ou moins intéressantes à ses propositions. L'agent a donc déjà envoyé une acceptation à la proposition qu'il a reçu et attend maintenant que son partenaire confirme ou annule leur mariage.
- **faithful** est l'état d'un agent qui est actuellement marié avec son partenaire.
- **unfaithful** est l'état d'un agent qui est actuellement marié, mais qui a reçu une proposition de la part d'un agent qu'il préfère à son partenaire actuel. L'agent a donc déjà envoyé une acceptation et attend alors une confirmation ou une annulation.

Pour communiquer avec les autres individus, chaque agent peut envoyer les messages suivants :

- **propose** permet de faire une proposition à un autre agent.
- **accept** (resp. **reject**) permet de répondre positivement (resp. négativement) à une proposition reçue d'un autre agent.
- **confirm** (resp. **withdraw**) permet de confirmer (resp. infirmer) un mariage, suite à une acceptation reçue.
- **divorce** permet de terminer un mariage.

Lors du déroulement de Casanova, chaque agent va donc communiquer avec les autres agents afin de progresser, et passera alors par trois phases : une phase de proposition, une phase de disposition, et une phase de remise en question.

Proposition. Au départ, chaque agent est libre (**free**) et enverra des propositions aux agent qu'il considère comme acceptables d'après son niveau de regret (un agent avec un niveau de regret k enverra une proposition aux k premiers partenaires dans sa liste de préférences). Lorsqu'un agent reçoit une réponse à toutes ses propositions, deux cas de figure se présentent. S'il n'a reçu aucune acceptation, l'agent augmentera son niveau de regret et enverra à nouveau des propositions. Si l'agent a reçu au moins une acceptation, il va alors choisir le partenaire qu'il préfère parmi tous ceux qui ont répondus positivement, et lui envoie un message de confirmation. L'agent envoie ensuite un message pour infirmer un mariage à chacun des autres agents qui lui aurait remis une acceptation. Pour finir, l'agent se considère comme marié (**faithful**) et passe en phase de remise en question. Lorsqu'un agent reçoit une proposition, il vérifie si elle est acceptable(i.e. si le partenaire est acceptable d'après son niveau de regret). Si cette proposition est acceptable, il passe en phase de disposition (**hesitating**), dans le cas contraire, il décline la proposition reçue.

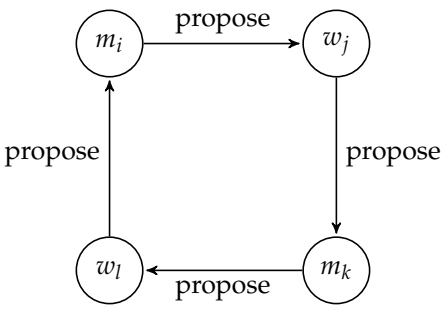
Disposition. Lorsqu'un agent reçoit une proposition acceptable alors qu'il est en attente de réponses, l'agent va hésiter (**hésitating**) avec ses propres propositions, qu'il a déjà envoyées. Il va donc attendre d'avoir reçu une

réponse à chacune de ses propositions avant de prendre une décision. Deux cas de figure peuvent se présenter suivant les réponses reçues. Si l'agent reçoit une acceptation d'une personne qu'il préfère a celle lui ayant envoyé une proposition, il va alors confirmer cette acceptation et se considérer comme marié (**faithful**), il va donc répondre négativement à la proposition qu'il a reçue. Dans le cas où l'agent n'aurait reçu aucune acceptation, ou dans le cas ou la proposition reçue est plus intéressante que chacune des acceptations, l'agent va alors accepter la proposition qu'il a reçu et se considère comme fiancé avec son partenaire (**engaged**). L'agent va alors maintenant attendre que son partenaire confirme le mariage et devenir marié (**faithful**), ou que son partenaire annule le mariage et devient alors à nouveau libre (**free**).

Remise en question. Une fois marié, un agent ne le reste pas forcement jusqu'à la fin de l'algorithme. En effet, si l'agent reçoit une proposition plus intéressante que son partenaire actuel, il va alors se considérer comme infidèle (**unfaithful**), et va considérer le proposant comme son amant avant de lui envoyer une acceptation. Si son amant confirme le mariage, l'agent va donc divorcer avec son partenaire actuel et va se marier avec son amant (**faithful**). Si son amant annule la proposition de mariage, l'agent redevient fidèle envers son partenaire actuel (**faithful**). Lorsqu'un agent marié reçoit un divorce, il va devenir libre(**free**) s'il n'a pas d'amant, et va devenir engagé (**engaged**) avec son amant dans le cas contraire et attendre une réponse en phase de disposition.

Arrêt. L'algorithme se termine lorsque tout les agents sont mariés, et la solution est l'appariement actuel. L'appariement ainsi obtenu est équitable envers les deux communautés puisque chaque individu a eu le même fonctionnement quelque soit sa communauté. De plus, grâce à l'approche distribué de l'algorithme, cet algorithme garantie la privacité de l'appariement puisqu'un agent ne connaît que son propre partenaire et n'a aucune indication sur les partenaires des autres agents.

Tel que l'algorithme est présenté dans [lien], il lui arrive dans de rares cas de se bloquer et donc de ne pas parvenir à fournir un appariement. Ce problème peut intervenir par exemple quand plusieurs agents sont bloqués entre-eux dans l'état hésitant. Puisqu'un agent hésitant attend d'avoir la réponse à toutes ses propositions avant de faire lui-même une réponse, un cycle de plusieurs agents hésitants peut se présenter dans lequel chaque agent attend une réponse de l'agent suivant pour pouvoir répondre à l'agent précédent, pour cette raison, l'algorithme n'atteint pas l'état de complétude. Puisque les agents ont le même comportement quelle que soit leur communauté, l'équité est atteint. Enfin, puisque le partenaire de chaque agent est connu de lui seul, la privacité est également atteint par Casanova.



Exemple 5. Exemple de situation d'interblocage dans Casanova.

2.6 Bilan

Maintenant que tous les algorithmes ont été présentés, nous allons donc revenir plus en détails sur leurs différents attributs, à savoir la complétude, la décentralisation, l'équité et la privacité :

Algorithme	Complétude	Décentralisation	Équité	Privacité
Gale Shapley	√			
Zig-Zag			√	
SML			√	
DisEGS	√	√		√
DisFC	√	√		√
Casanova		√	√	√

On notera tout d'abord que la distribution d'un algorithme semble être la solution pour lui permettre de conserver privé son appariement obtenue. On remarquera également qu'aucun des algorithmes présenté équitables ne parvient également à être complet. En effet, en dehors de l'approche simple issus de l'algorithme de Gale-Shapley, toutes les autres approches qui ont pour but de permettre de rendre un algorithme équitable entraîne des difficultés à garantir la stabilité des appariements obtenus. Pour finir, nous noterons que l'algorithme de Casanova est actuellement le seul capable de garantir à la fois la privacité et l'équité de son résultat. De plus, si les différents problèmes l'affectant actuellement et l'empêchant d'atteindre l'état de complétude étaient réglés, il serait alors la solution capable de répondre à toutes les exigences que l'on demande à un algorithme devant résoudre un problème SM. C'est ce que nous proposerons de faire par la suite.

3 Implémentation

Dans ce chapitre, nous rechercherons des outils d'évaluations nous permettant d'évaluer le bien-être de chaque individu, mais également le bien-être de communautés dans leurs ensembles. Grâce à ces différents outils, nous mettrons alors en place une démarche d'expérimentation, et présenterons les résultats obtenus afin de pouvoir mettre en lumière les différents points forts et points faibles de chaque algorithme. Nous reviendrons également sur le travail d'implémentation et de modification qui fut effectué sur certain algorithme afin de parvenir à mettre en place cette démarche expérimentale.

3.1 Critères d'évaluations

L'objectif que nous nous sommes fixés consiste à implémenter les différents algorithmes présentés dans le chapitre précédent afin de pouvoir les comparer. Pour cela il a été nécessaire de définir des critères sur lesquelles ces comparaisons pourraient s'effectuées, nous avons donc défini la notion de satisfaction individuelle.

Définition 2. *Soient un problème SM de taille n, ses deux communautés X et Y, et un individu z, et considérons un appariement M. On notera alors $\mu_M(z)$ le partenaire de z dans M. Si le rang de $\mu_M(z)$ dans la lise de préférence de z est k (avec $0 \leq k < n$). On définit alors la* ***fonction d'utilité*** *de l'individu z comme la fonction $u_z : X \cup Y \to [0, 1]$ définie telle que :*

$$u_z(\mu_M(z)) = \begin{cases} \dfrac{(n-1)-k}{n-1} & si\ \mu_M(z) \neq \emptyset \\ 0 & sinon \end{cases}$$

Maintenant que nous avons défini la notion de satisfaction individuelle, nous pouvons considéré le bien-être des individus dans leur ensemble, le bien-être d'une seul communauté ainsi que la notion de bien-être équitable qui prendra en compte l'équité d'une solution envers les deux communautés.

Définition 3. *Soient un problème SM de taille n, ses deux communautés X et Y, et un individu z, et considérons un appariement M.*

- *Le* ***bien-être utilitaire*** *considère la satisfaction de tous les individus : $sw_u(X \cup Y) = \frac{1}{2n} \sum_{z \in (X \cup Y)} u_z(\mu_M(z))$.*
- *Le* ***bien-être masculin*** *considère la satisfaction des hommes : $sw_u(X) = \frac{1}{n} \sum_{x \in X} u_x(\mu_M(x))$.*
- *Le* ***bien-être féminin*** *considère la satisfaction des femmes : $sw_u(Y) = \frac{1}{n} \sum_{y \in Y} u_y(\mu_M(y))$.*
- *Le* ***bien-être équitable*** *considère l'équité entre les hommes et les femmes : $sw_e(X \cup Y) = 1 - |sw_u(X) - sw_u(Y)|$.*

Notons que ces différentes notions de bien-être ont été normalisées (entre 0 et 1). Maintenant que ces notions sont bien définies, les critères qui ont été retenus pour comparer différents appariements sont, le bien-être utilitaire, le bien-être masculin, le bien-être féminin, le bien-être équitable, le nombre de couples bloquants présents, et le nombre de messages envoyés pour les algorithmes décentralisés.

La plupart des algorithmes étaient déjà implémentés en Java, les seuls manquants était DisFC et DisEGS. Pour implémenter ces deux algorithmes, nous avons décidé d'utiliser la plate-forme multi-agent Jason[1]. Jason est une version étendue du langage AgentSpeak[11] qui implémente la sémantique opérationnelle de ce langage.

3.2 Implémentation de DisEGS

Puisque DisEGS était l'algorithme le plus simple à implémenter des deux, nous avons décidé de commencer par celui-ci afin de bien prendre en main les différents outils mis à notre disposition par Jason. Grâce à l'algorithme fournie par Brito et Meseguer[3], il a suffit de traduire cet algorithme afin qu'il soit exploitable par Jason, de plus, la gestion d'envoi et de réception de messages a été grandement facilitée par sa simplification d'utilisation avec Jason.

Pour la terminaison de l'algorithme, une démarche différente est utilisée reposant sur la méthode des jetons de Dijkstra[6]. Pour cela, tous les agents sont triés suivant un ordre aléatoire, de sorte que chaque agent ne connaisse que sa position dans l'ordre et l'agent suivant. Ensuite, lorsqu'un agent est satisfait (dans notre situation, lorsqu'un agent est marié), il va vérifier sa position dans l'ordre. Si l'agent n'est pas le premier de l'ordre, rien ne se passe, en revanche, si c'est le premier, il va envoyer un jeton blanc (sous la forme d'un message) à l'agent suivant. Lorsqu'un agent reçoit un jeton blanc, deux cas de figure se présentent : si cet agent est également satisfait, il enverra un jeton blanc à l'agent suivant, dans le cas contraire, l'agent enverra un jeton noir à l'agent suivant. Si un agent reçoit un jeton noir, il se contentera de le faire passer à l'agent suivant. Lorsqu'un jeton est passé par tous les agents, le premier agent dans l'ordre va analyser la couleur du jeton qu'il aura reçu. Si ce jeton est blanc, cela signifie que tous les agents sont satisfaits et que l'algorithme peut se terminer. Dans ce cas, l'agent va alors contacter tous les autres agents pour leur demander d'écrire dans un nouveau fichier récapitulatif leurs noms ainsi que celui de leurs partenaires, avant de se terminer. Une fois que tous les autres agents ont été interrompus, le premier agent dans l'ordre termine l'exécution de l'algorithme. Dans le cas où le jeton reçu par l'agent d'ordre 0 est noir, cela signifie alors que tous les agents ne sont pas satisfaits et donc prêts à terminer l'algorithme. Si l'agent d'ordre 0 est encore satisfait au moment ou il reçoit ce jeton noir, il va tout de suite envoyer un nouveau jeton blanc à l'agent suivant. Si cet agent n'est plus satisfait à ce moment là, il va attendre d'être à nouveau satisfait avant d'envoyer un nouveau jeton.

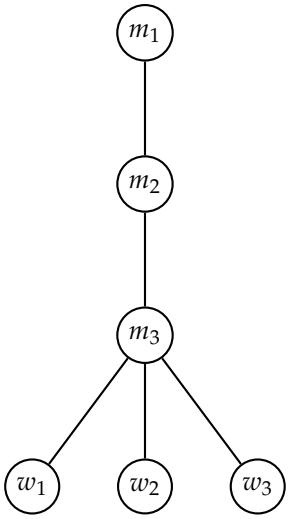
3.3 Implémentation de DisFC

Concernant l'algorithme DisFC, son implémentation a été plus longue puisque l'algorithme est plus important, de plus, certains points dans les algorithmes présentés n'étaient pas expliqués dans les détails ou ne faisaient que mention d'une heuristique à propos de la méthode utilisée sans expliquer ce qu'était cette heuristique. Concernant l'implémentation, certaines parties telles que la création de toutes les contraintes pour chaque agent, l'envoi d'informations par les hommes et leurs interprétations par les femmes, ont été des points dont l'implémentation

a été plutôt simple car expliquées en détail dans la publication. En revanche, le point qui a posé de nombreux problèmes a été les messages de *backtrack* que les femmes envoient aux hommes pour leur signaler qu'il n'y a plus de solution possible et qu'il leurs est nécessaire de choisir un autre partenaire. Pour être précis, la détection d'une situation problématique par une femme est simple à traiter, en revanche, le problème venait du choix de l'homme à contacter pour lui demander de changer de partenaire. La publication ne faisant que parler d'utiliser une heuristique sans l'expliquer, nous avons alors décidé d'utiliser une méthode proche de celle utilisé par l'algorithme Asynchronous Backtracking[12] qui permet de résoudre de façon distribué un CSP.

Dans l'algorithme Asynchronous Backtracking, il existe une relation d'ordre totale entre les agents. Ainsi, lorsqu'un agent a un problème à cause du choix de valeur de plusieurs agents de priorité supérieure, il va demander à l'agent de plus bas niveau parmi ces agents de priorité supérieure et lui demander de changer sa valeur. Si cet agent n'a alors à son tour plus aucune valeur à prendre, il va contacter l'agent de plus bas niveau parmi ses agents de priorité supérieure et lui demander de changer sa valeur. Ainsi, lorsqu'un blocage survient, l'information va remonter dans la hiérarchie jusqu'à trouver un agent qui puisse résoudre le problème (en prenant une autre valeur). Si suite à un blocage, l'information remonte jusqu'à l'agent de priorité maximale et que celui ci ne trouve aucune solution, cela signifie que toutes les solutions ont été tentées et qu'aucune ne permet de résoudre le CSP. L'agent abandonne alors la recherche de solution et termine l'algorithme. Dans le cas de DisFC, puisqu'il a été prouvé par Gale et Shapley grâce à leur algorithme qu'il existait toujours au moins une solution pour tout problème SM, cette situation d'abandon ne devrait jamais survenir.

Pour revenir à DisFC, la solution qui a été alors choisie fut de créer une relation d'ordre parmi les agents. Afin de permettre une plus grande clarté dans nos applications, nous allons utiliser l'analogie se basant sur la comparaison des individus avec des hommes et des femmes, telle que présenté dans GS, à savoir que le communauté composé des proposants sera celle des hommes et la communauté des disposants sera celle des femmes. Puisqu'il est indiqué que pour que l'algorithme fonctionnent correctement, les hommes doivent être les seuls à prendre des choix et que les femmes se contentent de concilier tous les choix qu'elles reçoivent, il a été décidé que dans tous les cas, un homme aura forcément une priorité supérieure à une femme. Puisque l'ordre entre les hommes n'a que peut d'importance, nous avons décidé d'utiliser les indices de ces hommes pour les trier. Ainsi, l'homme m_1 est l'homme de priorité maximale, alors que l'homme m_n est lui l'homme de priorité inférieure. Concernant la situation des femmes, puisqu'elles ne communiqueront jamais entre elles, il a été décidé de toutes les placer au même niveau de hiérarchie, situé au plus bas, juste après l'homme de priorité le plus faible.



Exemple 6. Exemple de la hiérarchie des agents pour une instance de taille 3.

Chaque agent connaît alors la liste de tous les agents de priorité inférieure, mais parmi les agent de priorité supérieure, il ne connaît que l'agent qui lui est directement supérieur dans la hiérarchie. Ainsi, lorsqu'une femme détecte un problème, elle envoie un message de backtrack à l'agent qui la précède dans la hiérarchie, à savoir l'homme de priorité minimale m_n. Lorsqu'un homme reçoit un message de backtrack, il va tout d'abord vérifier son correspondant est bien à jour concernant les valeurs qu'il a reçues, dans le cas contraire, il lui republiera sa valeur pour que l'agent se mette à jour. Si ce message de backtrack est pertinent, cet agent va alors se marier avec la femme suivante dans sa liste de préférence et prévenir de ce nouveau choix tous les agents de priorité inférieure. Si l'homme ne possède plus de partenaire potentiel dans sa liste de préférences, il va envoyer un message de backtrack à l'homme qui le précède pour lui demander de changer de partenaire, puis va lui-même se remarier avec sa partenaire préférée. De cette façon, une contradiction dans le choix d'un des hommes finira forcement par l'atteindre et il finira par changer son choix afin que l'algorithme trouve une solution.

Concernant la terminaison de DisFC, à nouveau la publication est plutôt floue sur la méthode à utiliser. Il a alors été décidé d'utiliser la même méthode que pour l'algorithme DisEGS, à savoir l'utilisation de jetons que les agents s'envoient pour signifier s'il sont satisfait ou non et permettre par la suite de stocker l'appariement obtenue dans un fichier.

4 Expérimentations

Dans ce chapitre, nous présentons le protocole expérimental que nous avons mis en place afin d'analyser les différents algorithmes présentés. Nous revenons également sur les différents problèmes qui se sont présenté lors de la mise en place de ce protocole ainsi que les modifications qui ont été effectuées afin de les résoudre. Pour finir, nous présentons et commentons les différentes courbes que nous avons obtenues grâce à cette expérimentation.

4.1 Évaluation des algorithmes

Afin d'évaluer les différents algorithmes implémentés, chaque algorithme est exécuté pour des instances de SM dont la taille est compris entre 2 et 100. Pour chaque taille de problème, 20 instances différentes sont lancées et on calcule la moyenne des résultats pour des instances de même tailles afin de pouvoir afficher par la suite ces résultats dans des courbes en fonction de la taille du problème SM. Le protocole expérimental étant lancé grâce au langage de programmation Java, il a fallu faire quelques préparations pour que des statistiques soit calculées à partir des algorithmes DisEGS et DisFC implémentés en Jason. Pour cela, nous avons implémenté en Java des générateurs permettant de transformer une instance de SM en programme Jason exécutable. Par la suite, ces programmes Jason sont exécutés afin de créer un fichier dans lequel l'appariement solution sera écrit. Une fois ce fichier créé, le programme de gestion d'expérimentations n'a plus qu'a le lire afin d'en extraire l'appariement solution, ainsi que les différentes statistiques tels que les différents bien-êtres ou le nombre de messages envoyés. Il suffit de répéter cette procédure pour toutes les instances de SM afin d'avoir des statistique pour DisEGS et DisFC.

4.2 Problèmes rencontrés

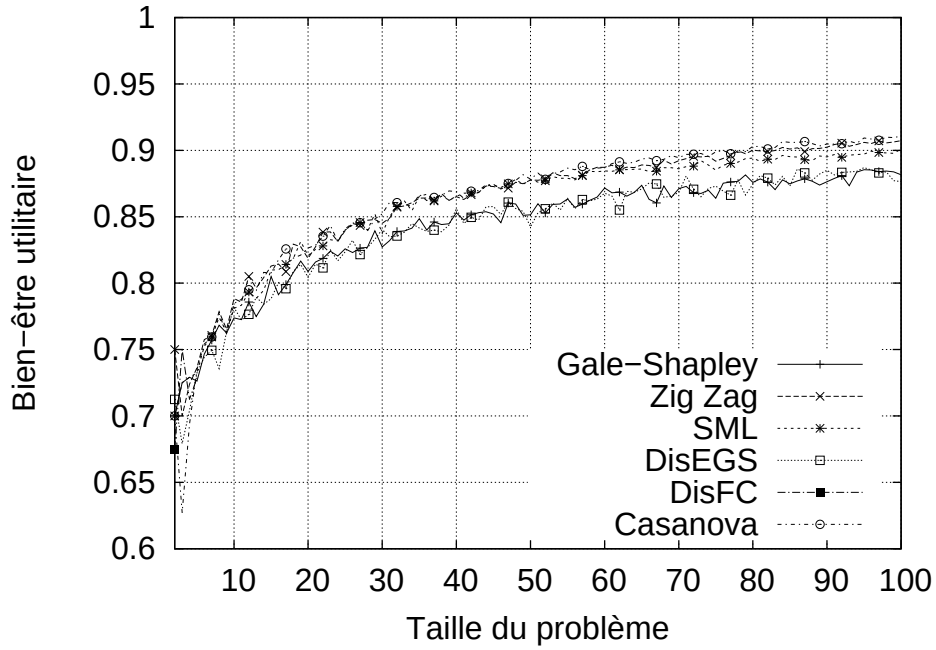
Une fois les premiers tests à grande échelle effectués sur les algorithmes DisEGS et DisFC effectués, nous avons réalisé que les appariements obtenus en résultat étaient dans de rare cas non stables, voir même incomplets. Après avoir étudié en détails les instances qui ont posé problème, nous avons remarqué qu'il y avait un souci avec la terminaison de nos algorithmes gérés par l'envoi de jetons. En effet, dans les instances ayant posé problème, l'algorithme se termine avant d'avoir obtenu un appariement correct. La condition permettant de garantir qu'un agent soit satisfait, à savoir qu'un agent soit actuellement marié ne convenait pas, puisque l'agent peut alterner entre l'état satisfait et non-satisfait rapidement et surtout fréquemment. De ce fait, lorsque l'agent de rang 0 dans l'envoi de jeton recevait, un jeton blanc, cela signifiait que tous les agents étaient satisfaits, mais ces agents ont pu entre temps recevoir des messages les rendant insatisfaits. Pour résoudre ce problème, la solution envisagée a été de maintenir l'utilisation des jetons, mais d'empêcher les agents de s'envoyer des jetons tant que tous les agents ne sont pas prêts à s'arrêter.

Dans un premier temps, il a été envisagé d'utiliser un agent superviseur afin d'analyser l'appariement actuel. En effet, à chaque fois qu'un agent décide de se marier avec un membre de l'autre communauté, il devait alors envoyer au superviseur son nom ainsi que celui de son conjoint afin que le superviseur puisse mettre à jour l'appariement. Lorsque le superviseur remarque que l'appariement actuel est stable, il ne lui restait plus qu'à envoyer un message à tous les agents afin de leur signaler qu'il peuvent à nouveau s'envoyer des jetons. Le problème de cette solution est principalement qu'elle remet en cause la privacité des algorithmes puisque qu'il existe au moins un agent qui connaît l'appariement dans son ensemble, de plus, la vérification qu'un appariement soit stable imposait que l'agent superviseur connaissent les listes de préférences de tous les individus.

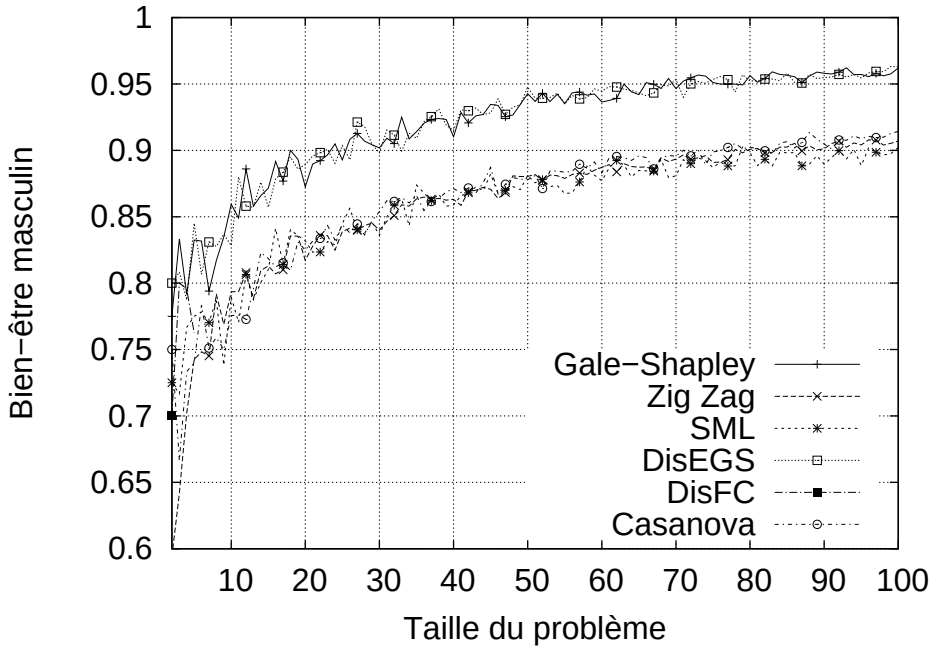
Une autre solution beaucoup plus simple nous est apparu après le constat suivant : quand un agent est satisfait, il n'envoie plus de message. Nous en avons déduis que lorsque tous les agents sont satisfaits, plus aucun agent n'envoie de message. Ainsi, pour détecter qu'un appariement stable est trouvé, il suffit de modifier l'agent superviseur afin qu'il surveille le flux de messages envoyés, sans avoir à vérifier le contenu des messages. Ainsi, lorsque le superviseur remarque que plus aucun message n'a été envoyé depuis un certain laps de temps, suffisamment long pour être significatif (ce laps de temps a été fixé en paramètre à intervalle de une seconde), le superviseur envoie alors un message à chaque agent pour lui indiquer qu'il peut envoyer des jetons à nouveau. Ainsi, avec cette méthode, la privacité n'est pas remise en cause puisque le superviseur ne connaît aucune information confidentielle sur les individus.

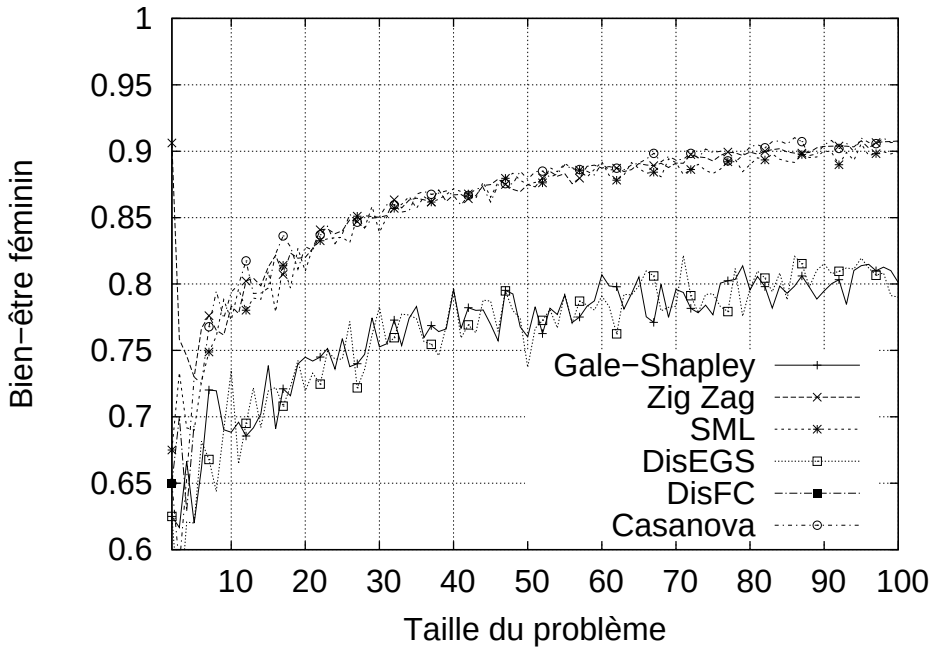
4.3 Analyse des résultats

Maintenant que des statistiques sont obtenues, nous pouvons comparer chaque algorithme. Il nous faut tout d'abord expliquer que l'expérimentation de DisFC n'a pas été effectuée jusqu'au bout. En effet, dès lors que la taille du problème était supérieure à 5, le temps de calcule nécessaire à l'exécution de 20 instances différentes comme le demande le protocole expérimental est trop important pour permettre à ce protocole d'arriver à son terme, comme cela pourrait se refléter dans la courbe représentant le nombre de messages envoyés par DisFC. Les résultats de DisFC sur les différentes courbes s'arrêtent donc dès les problèmes de taille supérieure à 5. Ce problème n'est pas dépendant de notre implémentation mais plutôt de algorithme lui-même puisque l'auteur obtient des résultats similaire concernant le nombre de messages envoyés[3].

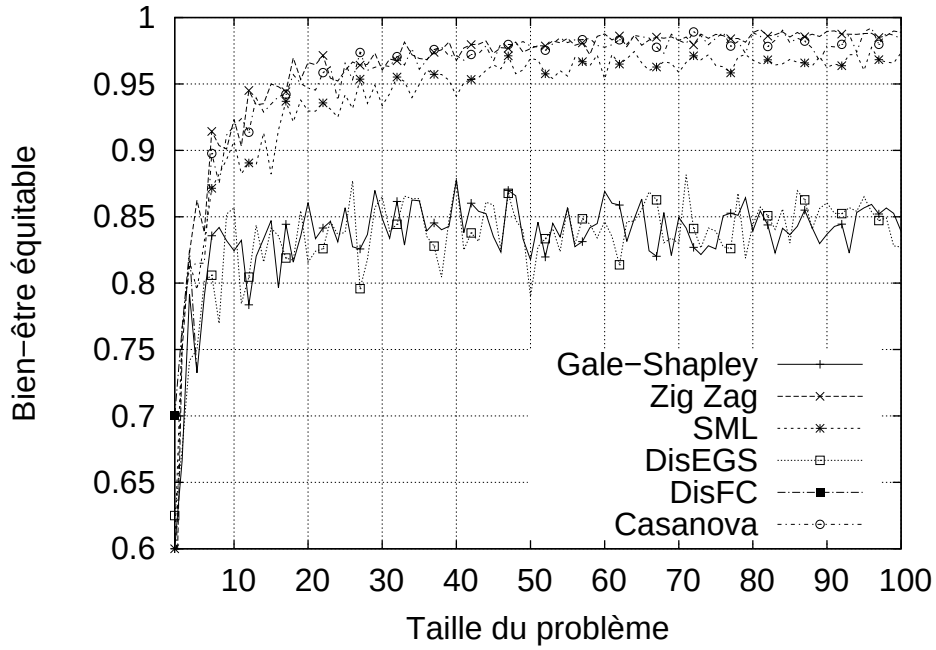


Concernant le bien-être utilitaire, nous remarquons que les algorithmes équitables obtiennent de meilleurs scores que les autres algorithmes, car bien que leur bien-être masculin soit légèrement plus faible, leur bien-être féminin est en revanche beaucoup plus élevé. On notera que parmi les algorithmes équitables, SML est légèrement moins performant que les deux autres. Enfin, malgré leurs approches presque identiques, les variations de résultats entre GS et DisEGS s'expliquent par un choix différent concernant l'ordre de passage des proposants.

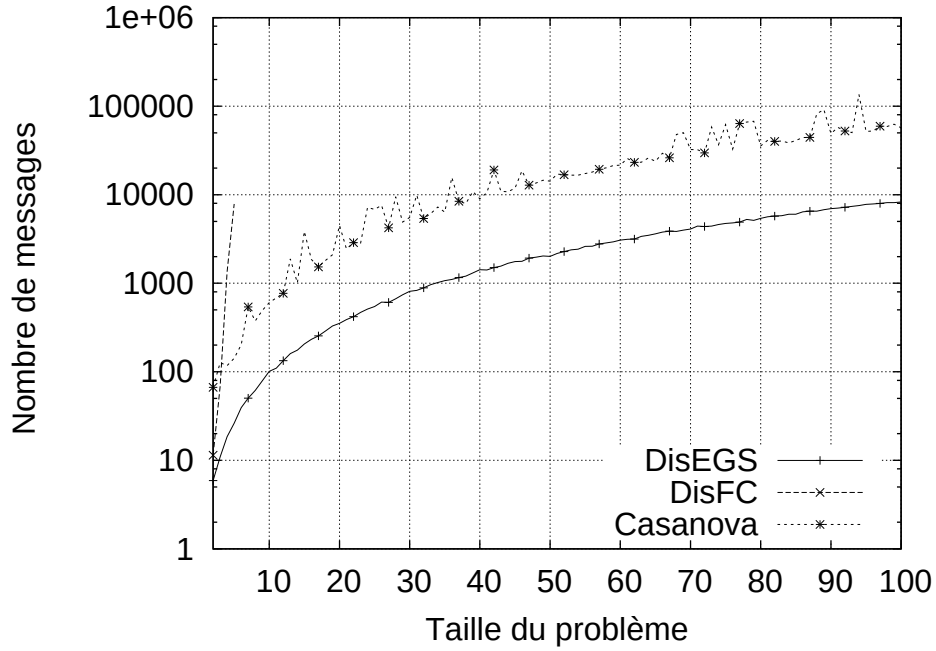




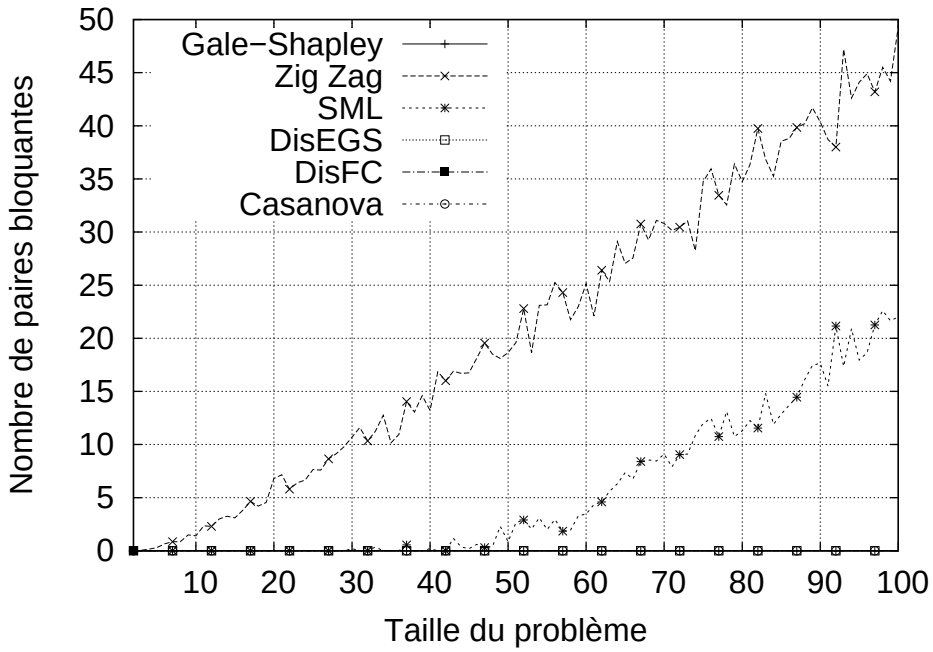
Concernant les bien-êtres masculin et féminin, les algorithmes équitables ont des résultats très proches pour ces deux métriques. Inversement, les autres algorithmes ont les meilleurs résultats concernant le bien-être masculin, mais également les plus mauvais en ce qui concerne le bien-être féminin.



Par extension, on remarquera de façon logique que les algorithmes équitables ont de très bon score en ce qui concerne le bien-être équitable. En revanche, les autres algorithmes favorisent grandement une communauté au détriment de l'autre, ce qui se reflète dans un score plus faible.



Parmi les différents algorithmes distribués, nous observons que DisEGS, ayant un fonctionnement très simple, est l'algorithme qui se termine le plus rapidement. Casanova en revanche, parvient à obtenir une solution équitable au pris d'un plus grand nombres de messages envoyés que DisEGS. DisFC pour sa part, à cause de sa complétude qui lui demande de tester un grand nombre d'appariements possibles, possède un nombre de messages envoyés extrêmement élevé, ce qui reflète le temps nécessaire trop important afin d'obtenir des résultats à partir de problème de taille 6.



Pour finir, concernant le nombre de paires bloquantes obtenue dans l'appariement solution, on notera que ZZ obtient les plus mauvais résultats, suivi par SML. Pour SML, les grandes variations sur les résultats obtenus pour des problèmes de tailles différentes s'expliquent par le choix aléatoire du premier appariement. Concernant Casanova, l'algorithme utilisé est celui utilisant les modifications proposées dans le chapitre suivant, ce qui lui permet de réduire à zéro son nombre de paires bloquantes obtenues.

5 Modification de Casanova

Maintenant que les différents algorithmes ont été analysés, nous avons cherché les différentes modifications que nous pourrions effectuer afin de leurs faire obtenir de nouvelles propriétés, ou de permettre à ces algorithmes de résoudre des problèmes proches ou des extensions d'un problème SM.

Dans un premier temps, nous nous sommes intéressés à l'algorithme DisFC, et avons plus particulièrement recherché un moyen de le rendre équitable entre les deux communautés. Pour cela, nous avons envisagé de modifier l'algorithme afin de permettre aux femmes puissent elles aussi faire des propositions et imposer leurs choix aux hommes. Pour cela, il faudrait modifier la hiérarchie des agents afin que les femmes ne soit pas dans tous les cas à un rang inférieur aux hommes. Ainsi, une solution serait de choisir aléatoirement la hiérarchie aux début de l'algorithme, une autre pourrait être de créer la hiérarchie de façon à alterner dans la hiérarchie les hommes et les femmes. Cependant, au vu de la lenteur d'exécution de DisFC, nous avons décidé d'abandonner cette idée et de nous concentrer plutôt sur les autres algorithmes.

Par la suite, nous avons étudié l'algorithme Casanova, et plus particulièrement la possibilité que l'algorithme puisse résoudre les extensions et problèmes proche du problème SM. Grâce à son approche dans laquelle le comportement des individus est identique quelque soit sa communauté, les modifications à apporter à l'algorithme serait sans doute minimes pour résoudre le problème des colocataires, dans lequel il n'existe qu'une seule communauté. Casanova pourrait être utilisé afin de résoudre un problème de SMI en modifiant la condition d'arrêt de l'algorithme. En effet, dans un problème de SMI, un individu n'est pas assuré de trouver un partenaire et peux être célibataire dans une solution au problème. Or, pour l'arrêt de Casanova, il faut que chaque agent soit satisfait, et donc marié, il faudrait donc modifier cette condition. Pour finir, puisque Casanova semble être l'algorithme qui offre le plus de possibilités, nous avons tout d'abord décidé de modifier l'algorithme afin de pouvoir résoudre les quelques rares cas dans lesquelles l'algorithme ne parvient pas à se terminer correctement, afin d'assurer sa complétude.

5.1 Implémentation de Casanova

La version de l'algorithme de Casanova sur laquelle nous avons travaillé a été implémentée en Java et simule un comportement multi-agent. Pour cela, une classe Java a été créé dans le but de représenté un agent, de façon à ce qu'un objet Agent soit créé pour chaque individu. Cette objet Agent contient toute les informations connues par un individu tel que son nom ou sa liste de préférence, et possède une boîte aux lettres dans laquelle seront stockés les messages reçus par un agent. Le déroulement de l'algorithme est alors légèrement modifié à cause de cette implémentation de la façon suivante : les agents ne travailleront pas tous en même temps, mais plutôt chacun leur

tour. Pour cela, le fonctionnement de l'algorithme repose sur des tours d'exécution de la façon suivante, à chaque tour, un ordre aléatoire est choisi parmi les agents. Ensuite le premier agent d'après cet ordre est sélectionné comme étant celui devant commencer à travailler. Lorsqu'un agent est choisi, il va consulter sa boîte aux lettres en commençant par les messages les plus anciens, et réagir comme s'il venait à l'instant de recevoir ce message. Ainsi, si un agent reçoit une proposition, il va interrompre la consultation de ses messages et va réagir à cette proposition en accord avec sa condition actuelle, une fois que l'agent a répondu au message de la façon approprié, il va supprimer le message reçu de sa boîte aux lettres et lire le message suivant. Lorsque l'agent a fini de consulter tous ses messages, il va se mettre en pause et attendre d'être sélectionné à nouveau. Pendant ce temps, l'agent suivant sera sélectionné afin qu'il commence à son tour à travailler. Une fois que tous les agents ont travaillé, le tour se termine et un nouvel ordre est choisi aléatoirement pour le tour suivant. À la fin de chaque tour, l'algorithme vérifiera si tous les agents sont actuellement satisfait afin d'arrêter l'algorithme et de retourner l'appariement solution. Afin d'éviter que l'algorithme se bloque lorsqu'il se retrouve dans une situation où plusieurs agents s'interbloquent, un nombre maximum de tours est fixé au lancement de l'algorithme en fonction de la taille du problème SM. Si ce nombre maximum de tours est atteint, l'exécution de l'algorithme est arrêtée de force et l'appariement actuel, potentiellement incomplet à cause d'un blocage, est sélectionné comme solution.

5.2 Problèmes rencontrés

Afin de résoudre les quelques cas posant problème, nous avons commencé par analyser ces cas et avons alors isolé deux situations différentes provoquant des résultats incorrects. Dans le premier cas, l'algorithme se bloque et l'appariement obtenu en résultat est incomplet, dans le deuxième cas, l'algorithme se termine normalement, mais l'appariement obtenu n'est pas stable.

5.2.1 Problème de stabilité des appariements

Concernant les appariements non stables obtenus, le problème fut rapidement identifié. Lorsqu'un agent est dans un des états suivants, à savoir, **hesitating**, **engaged**, ou **unfaithful**, cet agent va rejeter toutes les propositions qu'il reçoit tant qu'il n'atteint pas l'état **free** ou **faithful**. Or, prenons la situation suivante :
Soient m_i et w_j deux agents tels que m_i considère w_j comme un partenaire acceptable, de son coté, w_j n'est pas prêt à se marier avec m_i (son niveau de concession n'est pas assez élevé pour atteindre la position de m_i dans sa liste de préférence). m_i va alors envoyer une proposition à tous les partenaires qu'il juge acceptables, w_j pour sa part va refuser cet proposition qu'elle considère comme inacceptable. m_i peut alors atteindre l'état **engaged** avec un agent w_i qu'il apprécie moins que w_j mais ayant accepté sa proposition, et être bloqué dans cet état en attendant une confirmation ou une annulation du mariage. w_j va alors pour sa part finir par augmenter son niveau de concession suite à des rejets de tous ses partenaires potentiels et ainsi inclure m_i dans sa liste de partenaires potentiels. Or, lorsque w_j va faire ses propositions, m_i va rejeter toutes propositions de w_j reçues puisqu'il est actuellement dans l'état **engaged** avec w_k. w_j va alors concéder à nouveau et finir par accepter de se marier avec un partenaire m_l alors qu'elle aurait préféré être avec m_i qui a actuellement rejeté toutes ses propositions. À ce moment-là, w_k va enfin envoyer son message de confirmation et se marier avec m_i. Maintenant que m_i est dans l'état **faithful**, il serait prêt à accepter les propositions de w_j, mais celui-ci est actuellement marié et n'envoie plus de propositions. Ainsi, à la fin de l'algorithme, m_i sera marié avec w_k, et w_j sera marié avec m_l. Le couple (m_i, w_j) est alors un couple bloquant rendant l'appariement solution non stable. On notera alors que la situation aurait été

quasiment identique et aurait amené au même résultat si m_i avait été bloqué dans l'état **hesitating** ou **unfaithful**.

Pour résoudre ce problème, la première solution envisagée a été de modifier le fonctionnement des agents de façon à ce que lorsqu'un agent est dans l'état **hesitating**, **engaged**, ou **unfaithful**, cet agent accepte toutes les propositions si elles sont émises par un individu qu'il préfère à son partenaire potentiel actuel. Cet agent considère cet individu qu'il préfère comme son nouveau partenaire potentiel, rejette ensuite son ancien partenaire potentiel et continue son fonctionnement normalement. Cette solution a bien rempli son rôle, à savoir supprimer les couples bloquants, en revanche, cela a provoqué un nouveau problème. Lorsqu'un agent change de partenaire potentiel, il arrive que l'ancien partenaire potentiel ne parvienne pas à comprendre le rejet qu'il reçoit alors qu'il n'a lui même pas encore pris de décision définitive, et ignorera donc ce rejet. Cet agent pourra alors toujours se considérer comme marié comme s'il n'avait jamais reçu de rejet, ce qui provoquera à la fin de l'algorithme une situation ou deux agents se considèrent mariés avec la même personne, ce qui par la même occasion laissera un membre de la communauté opposée seul puisqu'il n'aura plus de partenaire célibataire prêt à l'accepter. Cette solution entraînerait des changements trop important et à la garantie incertaine, afin que ces nouveaux types de rejets soit bien compris.
Une autre solution fut proposée, consistant en la création d'un nouveau type de message. Lorsqu'un agent actuellement dans un état **hesitating**, **engaged** ou **unfaithful**, reçoit une proposition acceptable, il va lui envoyer un message de ce nouveau type nommé **postpone**, et continuer son processus comme s'il n'avait pas reçu cet proposition. Le proposant pour sa part, interprétera ce nouveau type de message de la façon suivante : le partenaire potentiel trouve la proposition intéressante, mais il n'est actuellement pas prêt à prendre une décision. Ainsi, lorsqu'un agent reçoit un message de ce type, il va simplement renvoyer une nouvelle proposition à son partenaire potentiel et ne pas changer son comportement actuel, à savoir attendre toutes les réponses à ses propositions pour prendre une décision. En expérimentant cette solution sur des instances particulières pour lesquelles Casanova ne parvenait pas à assurer la stabilité de la solution, nous avons constaté que les appariements obtenus était maintenant stables. Nous avons donc validé cette modification de l'algorithme et l'avons appliqué à la version de Casanova que nous utilisions jusqu'à présent.

5.2.2 Problème d'appariements incomplets

Concernant le problème de blocage de l'algorithme, qui entraînait un appariement incomplet en solution, l'origine est la suivante. Lorsqu'un agent est dans l'état **free** et en attente de réponse, s'il reçoit une proposition, cet agent passe dans l'état **hesitating**, considère le proposant comme un partenaire potentiel, mais n'envoie aucune réponse à cet agent. En effet, avant de pouvoir répondre à ce message, cet agent va attendre toutes les réponses à ses propres propositions afin de choisir le meilleur partenaire entre son partenaire potentiel et les agent ayant acceptés ses proposition. Ainsi, Lorsque deux agents qui se considèrent acceptables s'envoient mutuellement des propositions, chacun d'eux va entrer dans l'état **hesitating** et considérer l'autre comme partenaire potentiel. Cependant, aucun d'eux ne va répondre à l'autre puisque les deux agents attendent de recevoir leur réponse avant de répondre. On se retrouve donc dans une situation de blocage puisqu'aucun d'eux ne va céder et ces deux agents vont simplement attendre leur réponse jusqu'à ce que l'algorithme soit interrompu de force. Nous noterons que ce problème peut également impliqué un plus grand nombre d'agents en formant un cycle tel que présenté dans l'exemple 5. Afin de simplifier le raisonnement, nous nous intéresserons plus particulièrement au cas d'interblocage composé de seulement deux agents, chacune des solutions proposées pourra être étendue afin

de résoudre les situations impliquant un plus grand nombre d'agents. Pour résoudre ce problème, trois solutions ont été envisagées.

Pour commencer, la première solution proposée fut la suivante. Chaque message contient un paramètre permettant de dater un message, ainsi il est toujours possible de savoir chronologiquement parmi deux messages, lequel a été envoyé en premier. Ainsi, afin d'empêcher que deux agents ne se bloquent mutuellement, il a été décidé que lorsqu'un agent reçoit une proposition de la part d'un partenaire à qui il a déjà envoyé une proposition, deux cas de figures se présentent. Si cet agent a été le premier à faire une proposition chronologiquement parlant, il va alors rejeter la proposition reçue, dans le cas contraire, il considère la proposition comme une proposition normale. Ainsi, lorsque deux agents s'envoient des propositions mutuellement, le premier à avoir proposé rejettera le second, ainsi, le second aura une réponse à toutes ses propositions et pourra prendre la décision d'accepter la proposition issue du premier, ce qui finira par engendrer le mariage entre eux de ces deux agents. Or, après, avoir expérimenté cette méthode, nous avons remarqué que cela provoquait un problème dans le cas suivant :
Soient m_i et w_j deux agents tels que m_i considère w_j comme un partenaire acceptable, de son coté, w_j n'est pas prêt à se marier avec m_i. m_i va alors envoyer une proposition à tous les partenaires qu'il juge acceptables, w_j pour sa part va refuser cette proposition qu'il considère comme inacceptable. Suite aux rejets à ses propres propositions, w_j finira par concéder, m_i devint alors un partenaire acceptable, et w_j envoit alors une proposition à tous ses partenaires acceptables, m_i inclus. m_i pour sa part, en recevant cette proposition, remarquera alors qu'il a lui même effectué une proposition à ce même agent auparavant, et va rejeter cette proposition pour éviter une situation de blocage. w_j pour sa part, va donc décider de se marier avec quelqu'un qu'il apprécie moins puisqu'il a été rejeté par m_i. m_i de son côté, va alors choisir son partenaire préféré parmi les acceptants, et va contrairement à ce qu'il avait prévu, ne pas trouver d'acceptation de la part de w_j et va alors se marier afin quelqu'un qu'il apprécie moins mais qui a répondu favorablement à sa proposition. Ainsi, m_i et w_j ne seront pas mariés l'un avec l'autre et l'appariement ainsi obtenu ne sera donc pas stable.

La deuxième solution proposée se repose sur un autre constat : le problème actuel provient en grande partie de l'utilisation des boîtes aux lettres. En effet, dans une situation totalement distribuée, deux agents ne pourraient pas s'envoyer chacun une proposition acceptable, le premier à faire une proposition recevrait un rejet puisque le second ne serait pas encore prêt, de son coté, le second finira par concéder et inclure le premier comme partenaire acceptable et lui faire une proposition. À ce moment-là, le premier agent trouverait cette proposition acceptable, l'accepterait et ces deux agents finiraient par se marier ensemble. En revanche, avec l'implémentation en Java et son système de boîte aux lettres, en considérant le problème tel qu'il est énoncé précédemment, à savoir lorsque deux agents s'envoient une proposition en même temps, le déroulement se présentera de la façon suivante. Le premier agent à faire une proposition le fera parce qu'il sera le premier à agir, et fonctionnera de façon normal. En revanche, le second fonctionne de la façon suivante : lorsqu'il est au tour de cet agent de fonctionner, il va consulter sa boîte aux lettres de façon chronologique. Ainsi, s'il ne considérait pas encore comme acceptable le premier agent et que le premier message qu'il consulte soit le dernier rejet qu'il lui manquait, il va alors interpréter ce message de façon normal, à savoir, puisque cet agent n'a reçu aucune proposition acceptable, il va alors augmenter son niveau de concession, ce qui va rendre le premier agent acceptable pour le second. À ce moment précis le second agent va donc faire une proposition à tous ses partenaires qu'il considère comme acceptables, ce qui inclut le premier agent. En continuant par la suite à consulter ses messages, il va alors trouver la proposition du

premier agent et va l'accepter puisque actuellement, il est un partenaire acceptable, ce qui va provoquer l'interblocage. Or, à l'heure exacte où cette première proposition a été envoyée, le second agent ne trouvait en réalité le premier agent non acceptable, ce qui aurait provoqué un rejet de cette proposition dans une situation totalement distribuée, comme montré plus haut.
Ainsi, la solution proposée est la suivante : lorsqu'un agent est dans une situation dans laquelle il devrait faire ses propositions, il va tout d'abord, lire tous les messages restants dans sa boîte aux lettres avant de le faire. Ainsi, si une proposition intéressante est trouvé à ce moment, cet agent va accepter cette proposition et entrer dans l'état **engaged** avec le proposant, et ne fera donc pas ses propositions puisqu'il a trouvé un partenaire, ce qui empêchera la création d'une situation d'interblocage. Or, à nouveau cette solution provoquera la création de cas posant problème tel que celui-ci : soient m_i et w_j deux agents tel que m_i considère w_j comme un partenaire acceptable, de son coté, w_j n'est pas prêt à se marier avec m_i. m_i a donc accepté de se marier avec un autre agent w_k qu'il apprécie moins mais qui lui pour sa part était célibataire et prêt à se marier avec lui. w_j va par la suite recevoir le dernier rejet qu'il lui manquait et va alors augmenter son niveau de concession, ce qui inclura m_i dans la liste de ses partenaires acceptables. Or, avant d'effectuer ses propositions, il va vérifier le reste de sa boîte aux lettres et va trouver une proposition acceptable de la part d'un agent m_l qu'elle apprécie moins que m_i mais qui est malgré tout acceptable, elle va donc se marier avec m_l. Ainsi, m_i et w_j ne seront pas mariés l'un avec l'autre et l'appariement ainsi obtenue ne sera donc pas stable.

La troisième et dernière solution qui a été envisagée repose à nouveau sur un constat. Lorsqu'un agent est bloqué dans la situation **hesitating**, la seul réponse qu'il lui manque vient de la part de son partenaire potentiel qui lui a déjà envoyé une proposition. Ainsi, quand un agent se retrouve dans cette situation, il considère automatiquement que la réponse qu'il lui manque est une acceptation puisqu'il a déjà reçu une proposition de cette personne. Ainsi, cet agent va se considéré comme **engaged** avec ce partenaire et lui envoyer une acceptation. À nouveau, nous avons expérimenté cette solution sur les instances posant problème auparavant, et parvenons avec cette solution à obtenir un résultat stable et correct pour ces instances. Nous avons donc appliqué cette modification à notre version de l'algorithme de Casanova.

Pour finir, nous avons mis à l'épreuve notre nouvelle version de Casanova en lui faisant passer à nouveau le protocole expérimental proposé précédemment, à savoir exécuter cet algorithme sur 20 instances différentes pour des tailles de problème SM allant de 2 à 100. Nous avons alors constaté que toutes les solutions obtenues sont maintenant stables, en revanche, certaines exécutions de l'algorithme semble se bloquer et ne parviennent donc pas à fournir un appariement correct. Ce résultat reste malgré tout un progrès important puisque tous les appariements obtenus sont maintenant stables, de plus, le nombre de cas pour lesquels l'algorithme se bloquent à été réduit et est maintenant de l'ordre d'une dizaine de cas parmi les 2000 problèmes testés lors de notre démarche expérimentale. Pour finir, nous avons analysé les résultats de notre version de Casanova et les avons comparées aux autres algorithmes, ainsi, les courbes et résultats présentés dans le chapitre précédent reposent sur notre version de Casanova. Il est important de noter que telle qu'est définie notre approche expérimentale, puisqu'il est impossible d'extraire des résultats à partir des appariements incorrects obtenus, les statistiques obtenues le sont en se reposant uniquement sur les instances correctes.

Conclusion

Dans ce rapport, nous avons analysé le problème des mariages stables dans son ensemble, afin d'obtenir les résultats les plus pertinents possible. En effet, jusqu'à présent chaque algorithme avait ses propres critères d'évaluations et ne se comparaît qu'à GS. De ce fait, il était difficile de comparer entre-eux ces algorithmes afin d'en extraire leurs avantages et inconvénients, ou encore de savoir lequel utiliser afin de résoudre des cas concrets. Pour cela, nous avons défini différents critères, tels que l'équité, la privacité, ou encore la complétude, afin de savoir dans quelle situation un algorithme serait plus intéressant qu'un autre. De plus, nous avons défini différentes notions de bien-être afin de comparer différents appariements, et par extension pouvoir comparer tous les algorithmes dans des situations identiques. Pour finir, nous nous sommes intéressé plus particulièrement à l'algorithme de Casanova, et plus particulièrement à ses problèmes de complétude. Nous avons alors proposé des modifications afin de lui permettre de résoudre la majorité des cas qui lui pausait auparavant problème. Des progrès restent encore à effectuer sur cet algorithme, mais les différents atouts que possèdent actuellement Casanova nous permettent d'envisager son utilisation sur des problèmes plus larges que de simple SM, tels que le problème de mariage stable avec liste de préférences incomplètes, ou encore le problème des colocataires.

Le problème de mariage stable est un problème qui possède de nombreuses applications, allant de l'attribution de projets à des étudiants au covoiturage, en passant par les réseaux sociaux. Chacune de ces applications possèdent ses propres contraintes et à donc une approche qui lui conviendra le mieux. Ainsi, s'il est préférable de favoriser une communauté de plus grande importance au détriment de l'autre, l'algorithme le plus adapté ne sera pas le même que dans le cas contraire. De la même façon, une situation dans laquelle la confidentialité est indispensable imposera l'utilisation d'un algorithme respectant la privacité. Ainsi, même si Casanova est un algorithme très intéressant grâce à ses nombreuses propriétés, il existe toujours des situations dans lesquelles chaque algorithme est plus intéressant à utiliser qu'un autre.

Bibliographie

[1] R.H. Bordini, J.F. Hübner, and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak Using Jason*, John Wiley & Sons Ltd, Hoboken, New York, United States, 2007.

[2] I. Brito and P. Meseguer, 'Distributed forward checking', in *the Night International Conference on Principles and Practice of Constraint Programming*, pp. 801–806, (2003).

[3] I. Brito and P. Meseguer, 'Distributed stable matching problems', in *Principles and Practice of Constraint Programming - CP 2005*, ed., P. van Beek, volume 3709 of *Lecture Notes in Computer Science*, 152–166, Springer-Verlag, Heidelberg, Germany, (2005).

[4] I. Brito and P. Meseguer, 'Distributed stable matching problems with ties and incomplete lists', in *Principles and Practice of Constraint Programming - CP 2006*, ed., F. Benhamou, volume 4204 of *Lecture Notes in Computer Science*, 675–679, Springer-Verlag, Heidelberg, Germany, (2006).

[5] I. Brito and P. Meseguer, 'Distributed forward checking may lie for privacy', in *Recent Advances in Constraints*, eds., Francisco Azevedo, Pedro Barahona, François Fages, and Francesca Rossi, volume 4651 of *Lecture Notes in Computer Science*, 93–107, Springer-Verlag, Heidelberg, Germany, (2007).

[6] E.W. Dijkstra and C.S. Scholten, 'Termination detection for diffusing computations', *Information Processing Letters*, **11**(1), 1–4, (1980).

[7] P. Everaere, M. Morge, and G. Picard, 'Casanova : un comportement d'agent pour l'équité des mariages préservant la privacité', in *Actes des 19ème Journées Francophones sur les Systèmes Multi-Agents (JFSMA)*, pp. 191–200, Toulouse, France, (2011). Cepaduès.

[8] D. Gale and L. S. Shapley, 'College admissions and the stability of marriage', *American Mathematical Monthly*, **69**, 9–14, (1962).

[9] M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh, 'Local search algorithms on the stable marriage problem : Experimental studies', in *Proceedings of 19th European Conference on Artificial Intelligence (ECAI)*, pp. 1085–1086, Amsterdam, NL, (2010). IOS Press.

[10] D. Knuth, *Mariage stables*, Les Presses de l'Université de Montréal, Montréal, Canada, 1971.

[11] Anand S. Rao, 'AgentSpeak(L) : BDI agents speak out in a logical computable language', in *Proc. of MAAMAW*, (1996).

[12] M. Yokoo, E.H. Durfee, T. Ishida, and K. Kuwabara, 'The distributed constraint satisfaction problem : Formalization and algorithms', *IEEE Transactions on Knowledge and Data Engineering*, **10**, 673–685, (1998).

[13] B. Zavidovique, N. Suvonvorn, and G. S. Seetharaman, 'A novel representation and algorithms for (quasi) stable marriages', in *Proceedings of 2nd International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pp. 63–70, Barcelona, Spain, (2005). INSTICC Press.