MASTER IN COMPUTER SCIENCE

Stable Marriage and its application in master thesis assignment

Magnette, Loïc

Award date:
2013

Awarding institution:
University of Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Faculté d'Informatique
Année académique 2012–2013

Stable Marriage and its application in master thesis assignment

Loïc Magnette

Promoteur : Wim Vanhoof

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Table des matières

Remerciements

Je tiens, tout d'abord, à remercier monsieur Wim Vanhoof ainsi que tout le corps enseignant des Facultés Notre Dame de la Paix, pour l'aide, les conseils et le savoir qu'ils ont pu m'apporter dans le cadre de la réalisation de ce mémoire ainsi que pour la communication et le partage de leur savoir dans le cadre de mes études.

Je souhaite également remercier ma compagne, Justine Deghelt, mes parents, ma soeur, ainsi que tous mes proches qui m'ont soutenu durant ces deux années d'étude. Ils ont su m'encourager continuellement et me soutenir dans les quelques moments de faiblesse.

Je souhaite, enfin, remercier mon employeur, la société Oniryx, plus particulièrement Messieurs Geoffroy Fauveaux et Michaël Dewitte qui ont soutenu ma démarche d'apprentissage.

Chapitre 1

Introduction

L'algorithme du mariage a pour but de pouvoir créer/associer des couples sur base des éléments de 2 ensembles de tailles égales et ayant définis des préférences pour les éléments de l'autre ensemble. Cet algorithme garantit de créer un ensemble de couples et ce de manière optimale.

Un tel algorithme est fort pratique d'utilisation car bien que le problème de base soit la création de couple homme-femme, il peut etre appliqué à un large panel de problèmes connus (assignation de résidents à des hopitaux, création de colocations,...)

Un algorithme de ce type est utilisé au sein des Facultés Notre Dame de la Paix de Namur afin de proposer des assignations de mémoires sur base d'une liste de préférences fournies par les étudiants ainsi que sur base de leur performance académique.

Dans ce document, nous tenterons d'en apprendre un peu plus sur cet algorithme et aussi d'en déterminer sa complexité. Pour ce faire, nous réaliserons un rapide tour d'horizon des différentes formes de mariage stable et nous présenterons, ensuite, l'algorithme utilisé. Nous tenterons alors de d'évaluer sa complexité en le catégorisant.

Chapitre 2

Topologie de l'algorithme du mariage stable

2.1 Introduction

Cette partie a pour but de faire un rapide état des lieux des différentes formes de mariages stables connus. Cette itération ne sera pas exhaustive et se contentera d'énoncer les caractéristiques de ces versions et éventuellement leur(s) complexité(s). Pour ce faire, nous sommes partis d'un document publié [15] au sein de l'université de Glasgow.

2.2 Version classique

Pour commencer, nous allons décrire le concept classique de mariage stable[1]. Un tel problème est composé de deux ensembles disjoints, les hommes et les femmes et est dit de taille n. On notera, que la quantité totale de données pour une telle instance est en fait $O(n^2)$. Chaque personne définit une liste de préférence strictement ordonnée contenant les membres de l'autre ensemble. Ainsi une personne p préfère une personne q à personne r si et seulement si q précède r sur la liste de préférences de p.

Dans ce cadre, un couple M est une association un-à-un entre hommes et femmes. Pour un couple $(m, w) \in M$, on dit que m est le partenaire de w dans M, ou $m = pM(w)$, et de même pour w.

Il est ainsi possible de créer différentes attributions, c'est à dire des ensembles de couples tel que M. On distinguera deux types d'attributions opposées : stable et instable. On dira qu'une attribution (matching) est dite instable si elle admet une ou plusieurs paire(s) bloquante(s). Une paire bloquante pour un couple M est une paire homme-femme $(m, w) \notin M$, où chaque membre de cette paire préfère une autre personne à leur partenaire dans M [6, 5]. Au contraire, un matching sera dit stable s'il n'admet pas de paire bloquante.

1. SM : Stable marriage

Exemple :

Considérons l'ensemble M des hommes et leurs préférences :
- Tom : ⟨ *Emma, Kristen, Nina, Jude* ⟩
- John :⟨ *Kristen, Emma, Nina, Jude* ⟩
- Alex :⟨ *Nina, Jude, Kristen, Emma* ⟩
- Harry : ⟨ *Kristen, Jude, Nina, Emma* ⟩

Considérons l'ensemble W des femmes et leurs préférences :
- Jude : ⟨ *Alex, Tom, John, Harry* ⟩
- Nina : ⟨ *Tom, Harry, Alex, John* ⟩
- Kristen : ⟨ *Alex, John, Tom, Harry* ⟩
- Emma : ⟨ *Tom, John, Alex, Harry* ⟩

Sur base de ces listes de préférences, l'attribution suivante peut être crée :
⟨ { *Tom, Jude* {, *John, Nina* }, { *Alex, Kristen* }, { *Harry, Emma* } ⟩

Cette attribution n'est pas stable car elle admet au moins une paire bloquante, par exemple { *Tom, Emma* }.

Toujous sur base de ces listes de préférences, l'attribution suivante peut être crée :
⟨ { *Tom, Emma* }, { *John, Kristen* }, { *Alex, Nina* }, { *Harry, Jude* } ⟩
Cette attribution contrairement à la première est dite stable.

Il est relativement aisé de vérifier si une attribution M est stable. Il faut pour chaque homme m, vérifier que pour chaque femme qu'il préfère à sa partenaire dans M, cette femme w préfère m à son partenaire dans M. Si c'est le cas, la paire trouvée est dite bloquante. Par couséquent, l'attribution M est dite instable. A contrario, si aucune paire bloquante n'est trouvée on dira de cette attribution qu'elle est stable. Cette vérification peut être réalisée en un temps linéaire dépendant de l'entrée, et ce en utilisant des structures de données adaptées pour représenter les préférences.

Au travers de leur étude du mariage stable classique, Gale et Shapley on dévelopé un algorithme [2] permettant de trouver une attribution stable[4]. L'algorithme s'éxécute en $O(n^2)$ pour un exemple de taille n. Ils ont également démontré que pour toute instance du problème de mariage stable classique, il existe au moins une solution.[2].
Ils ont constaté que l'algorithme, qui exploite les propositions faites par les hommes aux femmes, produit une correspondance où chaque homme a la meilleur partenaire qu'il peut avoir dans toutes les attributions stables. On nomme cette attribution "men-optimal", on dira donc que l'algorithme est "men-oriented". En

2. Connu sous le nom de d'algorithme de Gale/Shapley (GS)

inversant les roles des hommes et des femmes ont obtient une solution "women-oriented" produisant des attributions "women-optimal"[4].

Plus tard, il a été constaté qu'une solution "men-optimal" se fait toujours au détriment des femmes. Plus précisément, chaque femme se voit associer le pire partenaire qu'elle puisse avoir parmis toutes les possibilités [15].
L'algorithme contient un élément de non-déterminisme, en effet, l'ordre dans lequel les hommes choisissent n'est pas donné.

On notera, que comme Gusfield et Irving l'ont montré [4], quel que soit l'ordre des propositions, le résultat obtenu sera toujours identique.

Une extension de l'algorithme de Gale-Shapley permet de simplifier le processus de création de paire. Cette version ajoute la suppression des hommes m' suivants m et ce dans l'ordre de préférence de la liste d'une femme w en faveur d'un homme m dont elle a reçu une proposition d'association. On supprime également w des listes de préférences de m'. Tous les couples stables sont contenus dans les GS-listes. Elles sont obtenues en prenant l'intersection des listes de préférences après l'application séparée de l'algorithme dans sa version "men-oriented" et "woman-oriented".[4]

2.3 Ensembles inégaux

Il est assez aisé d'envisager une forme dérivée de mariage stable classique dans laquelle les ensembles M et W sont de tailles inégales. Dans ce cas, il est évident que, quelque soit l'ensemble d'attributions, certaines personnes doivent se retrouver seules.

Comme nous l'avons vu précédement, afin de définir si une attribution est dite stable, il est nécessaire de définir le concept de paire bloquante. Ce concept doit être logiquement étendu pour couvrir ce cas. Ainsi une paire $(m, w) \notin M$ bloque M si m est soit sans partenaire, ou préfère w à sa partenaire dans M, et que w est soit sans partenaire, ou préfère m à son partenaire dans M [15]. Ci-après une illustration de cette forme dérivée de mariage stable et du concept de paire bloquante.

Exemple :

Considérons l'ensemble M des hommes et leurs préférences :
- Tom : ⟨ *Emma, Kristen, Nina, Jude* ⟩
- John :⟨ *Kristen, Emma, Nina, Jude* ⟩
- Alex :⟨ *Nina, Jude, Kristen, Emma* ⟩
- Harry : ⟨ *Kristen, Jude, Nina, Emma* ⟩
- Jake : ⟨ *Emma, Kristen, Nina, Jude* ⟩

Considérons l'ensemble W des femmes et leurs préférences :
- Jude : ⟨ *Alex, Tom, John, Harry, Jake* ⟩
- Nina : ⟨ *Tom, Harry, Alex, John, Jake* ⟩
- Kristen : ⟨ *Alex, John, Tom, Harry, Jake* ⟩
- Emma : ⟨ *Tom, John, Alex, Harry, Jake* ⟩

Sur base de ces listes de préférences, l'attribution suivante peut être crée :
⟨ { *Tom, Jude*}, { *John, Nina* }, { *Jake, Kristen* }, { *Harry, Emma* } ⟩.

Cette attribution n'est pas stable car elle admet au moins une paire bloquante, par exemple { *Tom, Emma* } comme pour le cas classique. Une autre paire bloquante intérresante peut être envisagée $\{John, Kristen\}$. Dans l'attribution proposée *John* est sans partenaire. Une telle paire n'existe pas dans le problème de mariage stable classique.

Toujous sur base de ces listes de préférences, l'attribution suivante peut être crée :
⟨ { *Tom, Emma*}, { *John, Kristen* }, { *Alex, Nina* }, { *Harry, Jude* } ⟩
. Cette attribution contrairement à la première est dite stable.

Nous avons vu que dans le cas d'un mariage stable classique, il existe toujours au moins une attribution stable comprenant tout les membres des deux ensembles. Cette propriété n'est plus exacte dans le cadre des ensembles de tailles inégales. Cependant, cette propriété peut être dérivée en une forme différente dans notre cas. En effet, si on considère l'ensemble le plus petit, il apparaît de manière évidente que cette propriété est toujours vrai pour cet ensemble. Ainsi, on peut dire qu'il existe au moins une attribution stable dans laquelle tous les membres de l'ensemble le plus petit sont mis en correspondance.

Il a été montré [12] que le plus grand ensemble est divisé en deux sous-ensembles, les membres ayant un partenaire dans toutes les associations stables et les autres.

2.4 Liste incomplète

Une forme fréquente de mariage stable est celle où les listes de préférences incomplètes sont autorisées. Cette forme est habituellement notée SMI ("Stable marriage with incomplete list"). Le principe est le suivant, lorsque une personne veut marquer un membre de l'autre ensemble comme inadéquat, elle l'omettra de sa liste de préférences.

Comme nous l'avons fait pour les ensembles de tailles inégales, il est nécessaire de revoir le concept de stabilité. Pour ce faire il faut revoir la définition de paire bloquante. Considérons A l'ensemble des paires acceptables dans une instance I de SMI et a égal à $|A|$. Une association M est un sous-ensemble des paires acceptables de A telle que $|\{m : (m, w) \in M\}| \leq 1$ pour tout m et

$|\{w : (m, w) \in M\}| \leq 1$ pour tout w. Une paire $(m, w) \in M$ est bloquante si m est soit célibataire ou préfère w à sa partenaire dans M, et w est soit célibataire ou préfère m à son partenaire dans M. Une attribution stable est donc une attribution ne contenant pas de paire bloquante.

Exemple :

Considérons l'ensemble M des hommes et leurs préférences incomplètes :
- Tom : ⟨ *Emma, Kristen, Nina* ⟩
- John :⟨ *Kristen, Emma, Nina* ⟩
- Alex :⟨ *Nina, Jude, Kristen* ⟩
- Harry : ⟨ *Kristen, Jude* ⟩

Considérons l'ensemble W des femmes et leurs préférences incomplètes :
- Jude : ⟨ *Tom, Harry, Alex* ⟩
- Nina : ⟨ *Tom, Harry, Alex* ⟩
- Kristen : ⟨ *Alex, John, Tom* ⟩
- Emma : ⟨ *Tom, John, Alex* ⟩

Sur base de ces listes de préférences, l'attribution suivante peut être crée :
⟨ { *Tom, Kristen*}, { *John, Emma* }, { *Alex, Jude* } ⟩

Cette attribution n'est pas stable car elle admet au moins une paire bloquante, par exemple { *Tom, Emma* } comme pour le cas classique. Une autre paire bloquante intérresante peut être envisagée $\{Harry, Kristen\}$. Dans l'attribution proposée *Harry* est sans partenaire. Une telle paire n'existe pas dans le problème de mariage stable classique.

Une conséquence évidente de cette extension est que certaines attributions seront impossibles. Un exemple simple est qu'il peut exister des personnes n'ayant aucune correspondance parmi leur liste de préférences.

Exemple :

Considérons l'ensemble M des hommes et leurs préférences :
- Tom : ⟨ *Emma, Kristen, Nina, Jude* ⟩
- John :⟨ *Kristen, Emma, Nina, Jude* ⟩
- Alex :⟨ *Nina, Jude, Kristen, Emma* ⟩
- Harry : ⟨ *Kristen, Jude, Nina, Emma* ⟩

Considérons l'ensemble W des femmes et leurs préférences :
- Jude : ⟨ *Alex, Tom, John* ⟩
- Nina : ⟨ *Tom, Alex, John* ⟩
- Kristen : ⟨ *Alex, John, Tom* ⟩
- Emma : ⟨ *Tom, John, Alex* ⟩

Il apparait clairement dans l'exemple précédent que $Harry$ ne dispose d'aucune correspondance dans les listes de préférences des femmes.

Dans ce cadre, on associera une notion de cohérence aux listes de préférences [15]. Une liste de préférences sera dite cohérente si un individu i apparaît dans la liste de préférences d'un second individu j , si et seulement si, j apparaît dans la liste de i. Par la suite on présumera toujours de la cohérence des listes de préférences. Toute liste peut être rendue cohérente en un temps linéaire en rapport avec la taille des entrées. Ceci est bien sûr fait sans changer l'ensemble des couples stables car, pour rendre une liste cohérente, on supprimera des préférences qui ne feront jamais apparaître de couple; la préférence étant unilatéral. Tenant compte de la cohérence des listes de préférences, on considère le couple homme m et la femme w comme acceptable si chacun apparaît dans la liste de préférences de l'autre [15].

Il a été montré que dans une instance de SMI, les deux ensembles hommes et femmes sont chacuns divisés en deux sous-ensembles [8]. Ces sous-ensembles englobent les individus qui ont des partenaires dans toutes les attributions stables ainsi que ceux n'ayant de partenaires dans aucune attribution stables [10].

Dans le cas de séries de tailles inégales, une instance de SMI peut avoir le même ensemble d'association qu'une occurence SM [15]. Pour ce faire il suffit de suivre le processus suivant :

- supprimer les paires non appariées
- ensuite, pour chaque personne p restante, ajouter toutes les autres du sexe opposé qui n'apparaissent pas déjà dans la liste de p

On utilise généralement a comme mesure de complexité en temps, tel que a est le nombre de couples acceptables pour une instance du problème.

2.5 Indifférence

Le problème du mariage stable classique requiert à chaque membre des deux ensembles de fournir une liste de préférence dans laquelle les personnes sont classées de manière stricte (aucune égalité). Cependant, il est assez naturel d'envisager la possibilité d'indifférence voir d'égalité entre deux personnes. Cette possibilité est une extension du problème classique qui ouvre à une variété de problèmes et de résultats intéressants.

L'indifférence peut amener à diverses interprétations. La plus naturelle se traduit par l'existence d'égalité. Plus formellement, il faut envisager un ensemble W de k femmes étant à égalité dans la liste de préférences d'un homme m, de manière telle que m n'a pas de préférence entre wi et wj, $\forall wi, wj \in W$. Alors que, $\forall w \notin W$ et appartenant à la liste de préférence de m, soit m préfère w à toute autre femme de W, soit m préfère toutes les femmes de W à w. Cette extension du problème proposent ainsi deux nouvelles formes de mariage stable : mariage stable avec égalités et listes complètes (SMT) et mariage stable avec égalité et listes incomplètes (SMTI).

Exemple :

Les égalités dans l'exemple suivant sont notées entre parenthèses. Considérons l'ensemble M des hommes et leurs préférences :

- Tom : ⟨ *Emma, Kristen, Nina, Jude* ⟩
- John :⟨ *Kristen, (Emma, Nina) , Jude* ⟩
- Alex :⟨ *Nina, Jude, Kristen, Emma* ⟩
- Harry : ⟨ *Kristen, Jude, Nina, Emma* ⟩

Considérons l'ensemble W des femmes et leurs préférences :

- Jude : ⟨ *Alex, Tom, John, Harry* ⟩
- Nina : ⟨ *Tom, Harry, Alex, John* ⟩
- Kristen : ⟨ *Alex, John, Tom, Harry* ⟩
- Emma : ⟨ *Tom, (John, Alex) , Harry* ⟩

Dans cette on peut ainsi constaté que par *John* n'a pas de préférence entre *Emma* et *Nina*. Il en va de même dans le cas d'*Emma* qui n'a pas de préférence entre *John* et *Alex*.

Il existe une relation assez simple entre le problème de mariage stable avec égalité (et listes incomplètes) et le problème de mariage stable (avec listes incomplètes). En effet, pour une instance I de SMT(I), on peut produire une ou plusieurs instances de SM(I), en forcant l'ordonnancement dans I, c'est à dire en rompant les égalités. Les intances produites par cette opération sont appelées instances dérivées.

Logiquement, si on autorise l'indifférence, il nécessaire de redéfinir la notion de stabilité. Pour ce faire nous devons réévaluer la notion de paire bloquante. Considérons une paire n'appartenant pas à une association comme bloquante pour une association si, de par leur coexistance, les deux associations :

- sont soit dans une meilleure situation sans,
- soit sont dans une situation pire sans,
- ou encore, une des associations est dans une meilleure situation sans tandis que la situation de l'autre est pire sans.

Ces trois possibilités donnent lieu à des notions de stabilité faible, super-stabilité et forte stabilité [11, 7]. Ces trois notions seront d'avantage illustrées par la suite.

2.5.1 Stabilité faible

Nous commencerons par étudier la forme de stabilité la plus faible. Pour ce faire, nous commencerons par définir la notion de paire bloquante dans ce cadre. Considérons une instance I de SMT. Une paire n'apparaissant pas dans M, bloque M si chaque membre de la paire préfère l'autre à son partenaire dans M [7]. On dira donc que M est faiblement stable, s'il n'existe pas de paire bloquante pour M.

Exemple :

Considérons l'ensemble M des hommes et leurs préférences :
- Tom : ⟨ *Emma, (Kristen, Nina), Jude* ⟩
- John :⟨ *Kristen, (Emma, Nina), Jude* ⟩
- Alex :⟨ *Nina, Jude, Kristen, Emma* ⟩
- Harry : ⟨ *Kristen, Jude, Nina, Emma* ⟩

Considérons l'ensemble W des femmes et leurs préférences :
- Jude : ⟨ *Alex, Tom, John, Harry* ⟩
- Nina : ⟨ *Tom, Harry, Alex, John* ⟩
- Kristen : ⟨ *Alex, John, Tom, Harry* ⟩
- Emma : ⟨ *Tom, John, Alex, Harry* ⟩

Sur base de ces listes de préférences, l'attribution suivante peut être crée :
⟨ { *Tom, Jude* }, { *John, Nina* }, { *Alex, Kristen* }, { *Harry, Emma* } ⟩

Cette attribution n'est pas stable car elle admet au moins une paire bloquante par exemple { *Tom, Emma* } comme pour le cas classique. Une autre paire intérresante à envisager est { John, Emma }. Cette paire n'est pas bloquante mais elle exploite le mécanisme d'indifférence.

De plus on dira que M est faiblement stable dans I, s'il est stable pour au moins une instance I' dérivée de I.

Le concept de stabilité faible est un concept difficile. On sait par exemple que des problèmes tels que trouver une attribution avec égalité faiblement stable ou trouver une attribution avec regret minimum faiblement stable sont tous deux NP-complet, même si les égalités ne sont présentes que d'un seul coté [13].

2.5.2 Forte stabilité

Nous venons de voir la forme la plus faible de stabilité, nous allons maintenant nous intéresser à une forme intermédiaire ; la notion de forte stabilité. Pour commencer nous allons redéfinir, dans ce contexte, le concept de paire bloquante et ainsi redéfinir également la notion de stabilité. Considérons une instance I de SMT. Une paire n'apparaissant pas dans M, bloque M si un membre de la paire préfère l'autre à son partenaire dans M alors que l'autre est au moins indifférent [10]. M est dit fortement stable s'il n'existe pas une telle paire bloquante pour M.

Exemple :

Considérons l'ensemble M des hommes et leurs préférences :
- Tom : ⟨ *Emma, Kristen, Nina, Jude* ⟩
- John :⟨ *Kristen, (Emma, Nina), Jude* ⟩
- Alex :⟨ *Nina, Jude, Kristen, Emma* ⟩
- Harry : ⟨ *Kristen, Jude, Nina, Emma* ⟩

Considérons l'ensemble W des femmes et leurs préférences :
- Jude : ⟨ *Alex, Tom, John, Harry* ⟩
- Nina : ⟨ *Tom, Harry, Alex, John* ⟩
- Kristen : ⟨ *Alex, John, Tom, Harry* ⟩
- Emma : ⟨ *Tom, John, Alex, Harry* ⟩

Sur base de ces listes de préférences, l'attribution suivante peut être crée :
⟨ { *Tom, Jude* }, {, *John, Nina* }, { *Alex, Kristen* }, { *Harry, Emma* } ⟩

Cette attribution n'est pas stable car elle admet au moins une paire bloquante par exemple { *Tom, Emma* } comme pour le cas classique. La paire { John, Emma } est également bloquante dans ce cas alors qu'elle ne l'est pas dans un cas de stabilité faible. En effet, *Emma* préfère *John* à *Harry* tandis que *John* n'a pas émis de préférence entre *Emma* et *Nina*.

Une définition alternative, peut être formulée comme ceci [15] : Un matching M est fortement stable dans une instance I de la SMT, si et seulement si,

1. il existe une instance de SMT I' obtenue à partir de I par la suppression des égalités du côté des hommes, de sorte que pour chaque instance de SM obtenue à partir de I' en supprimant les égalités (du côté des femmes), M est stable
2. et s'il existe une instance I' du SMT obtenue à partir de I par la suppression des égalités du côté des femmes, de sorte que pour chaque instance de SM obtenue à partir de I' en supprimant les égalités (du côté des hommes), M est stable.

On peut aisément envisager une instance de SMT, pour laquelle une attribution fortement stable ne peut exister. Considérons le cas de SMT, illustré dans la figure ci-après. Les personnes à égalité dans une liste de préférences sont notées entre parenthèses (nous utiliserons cette notation tout au long de ce document). L'attribution $(m1, w1), (m2, w2)$ est bloquée par $(m2, w1)$, tandis que l'attribution $(m1, w2), (m2, w1)$ est bloquée par $(m2, w2)$. Il existe cependant un algorithme $O(n^4)$ qui, pour une instance de SMT avec n hommes et n femmes, détermine si une telle attribution existe, et si tel est le cas en trouver une [6].

$m1 : w1w2$ $\quad w1 : m2m1$
$m2 : (w1w2)$ $\quad w2 : m2m1$

2.5.3 Super stabilité

Nous venons de voir les notions de stabilité faible et forte, nous allons maintenant passer à la plus forte notion de stabilité soit la super stabilité. Nous allons commencer par définir la notion de stablité et pour ce faire la notion de paire

bloquante. Soit I une instance de SMT. On dira qu'une paire qui n'apparaît pas dans M, bloque M si chacun de ses membres préfère l'autre à son partenaire dans M, ou si les membres sont indifférents entre eux [7, 10]. Logiquement on dira donc que M est super-stable si une telle paire n'existe pas.

Exemple :

Considérons l'ensemble M des hommes et leurs préférences :
- Tom : ⟨ *Emma, (Kristen, Nina), Jude* ⟩
- John :⟨ *Kristen, Emma, Nina, Jude* ⟩
- Alex :⟨ *Nina, Jude, Kristen, Emma* ⟩
- Harry : ⟨ *Kristen, Jude, Nina, Emma* ⟩

Considérons l'ensemble W des femmes et leurs préférences :
- Jude : ⟨ *Alex, Tom, John, Harry* ⟩
- Nina : ⟨ *Tom, Harry, Alex, John* ⟩
- Kristen : ⟨ *Alex, (John, Tom), Harry* ⟩
- Emma : ⟨ *Tom, John, Alex, Harry* ⟩

Sur base de ces listes de préférences, l'attribution suivante peut être crée :
⟨ { *Tom, Nina*}, { *John, Kristen* }, { *Alex, Jude* }, { *Harry, Emma* } ⟩

Cette attribution n'est pas stable car elle admet au moins une paire bloquante par exemple { *Tom, Emma* } comme pour le cas classique. La paire { Tom, Kristen } est également bloquante dans ce cas alors qu'elle ne l'est pas dans un cas de stabilité forte. En effet, *Tom* n'a pas défini de préférence ente *Kristen* et *Nina* et *Kristen* n'a pas défini de préférence ente *John* et *Tom*.

Une attribution est super-stable si elle est stable pour tout I' de SM qui peut être dérivée de I [15].

Il est possible qu'une instance de SMT n'admet aucune attribution super-stable. Ainsi, si pour une intance donnée I, toutes les listes de préférences ne contiennent aucune préférence stricte, il ne peut y avoir d'attribution super-stable. Il existe un algorithme en $O(n^2)$ qui détermine pour une instance I contenant n hommes et n femmes s'il existe une attribution super-stable et si tel est le cas en trouve une [6].

Nous avons jusqu'a présent interprété l'indifférence comme une égalité. Une autre interprétation est de considérer l'indifférence comme une information incomplète. Imaginons que l'on ne dispose pas de l'information concernant la préférence d'un homme m entre une femme $w1$ et une femme $w2$. Si, de ce fait, on met $w1$ et $w2$ en égalité dans la liste de préférences de m et que l'on trouve une attribution "super-stable". On peut, donc, en conclure que la préférence dans la liste de m n'a pas d'importance, vu que l'attribution sera stable pour

chaque instance de SM que l'on peut dériver de l'instance d'origine. De ce fait, les attributions "super-stable" sont des attributions où l'indifférence représente généralement une information incomplète [15].

Il est clair qu'une attribution super-stable est fortement stable, et une attribution fortement stable est aussi faiblement stable.

2.6 Autres formes de mariage stable

Précédement, nous avons présenté quelques grands types de mariage stable. Il est bien évident que celà ne rassemble qu'une partie des problèmes existants. Davantage de références sont disponibles dans la littérature[15]. On y retrouve entre autre des formes de mariage stable avec listes incomplètes et égalités.

Il nous semble, cependant, important d'évoquer une forme intéressante de mariage stable dans le cadre de ce document. Cette forme est nommée "Stable marriage with master list" (SM-ML) [14]. Elle peut coexister avec les différentes formes déjà vues précédemment. On définit une telle forme de la manière suivante. Une "master list"(ML) est une liste, pouvant contenir des égalités, définissant les préférences d'un ensemble A sur un ensemble B tel que pour $a_1, a_2 \in A$, $pref(a_1) = pref(a_2)$, où $pref(x)$ donne la liste de préférences de l'élément x. Il existe 2 formes de SM-ML :

- SM-1ML : dans ce cas, seul un des deux ensembles dispose d'une "master list"
- SM-2ML : dans ce cas, les deux ensembles disposent chacun d'une "master list"

Exemple :

Dans cette exemple nous envisagerons le cas de SM-1ML Considérons l'ensemble M des hommes et leurs préférences :

- Tom : ⟨ *Emma, Kristen, Nina, Jude* ⟩
- John :⟨ *Kristen, Emma, Nina, Jude* ⟩
- Alex :⟨ *Nina, Jude, Kristen, Emma* ⟩
- Harry : ⟨ *Kristen, Jude, Nina, Emma* ⟩

Considérons l'ensemble W des femmes et leurs préférences, la liste de préférences étant commune à toutes les femmes :

- Jude : ⟨ *Alex, Tom, John, Harry* ⟩
- Nina : ⟨ *Alex, Tom, John, Harry* ⟩
- Kristen : ⟨ *Alex, Tom, John, Harry* ⟩
- Emma : ⟨ *Alex, Tom, John, Harry* ⟩

Chapitre 3

Présentation de l'algorithme

Nous allons à présent nous intéresser à l'algorithme utilisé aux Facultés Notre Dame de la Paix de Namur. Dans ce but, nous commencerons par expliquer la problématique rencontrée. Nous tenterons ainsi de la formaliser. Dans un second temps nous présenterons l'algorithme et son fonctionnement. Pour ce faire, nous nous baserons sur un document rédigé par Wim Vanhoof [16].

3.1 La problématique

Nous allons commencer par décrire la problématique de manière générale, nous la formaliserons par la suite.

Les Facultés Notre Dame de la Paix proposent chaque année différents sujets de mémoire aux étudiants. Les étudiants peuvent ensuite faire part de leurs préférences parmi ces sujets. Pour ce faire, les étudiants doivent remettre une liste de préférences contenant un nombre limité de sujets. Les sujets sont ensuite attribués sur base des listes de préférences des étudiants ainsi que sur base de leurs résultats académiques. Le problème étant de produire une attribution stable. Nous définirons plus loin la notion de stabilité.

Considérons donc, un ensemble S contenant les étudiants souhaitant profiter des mémoires proposés. L'ensemble S est un ensemble fini partiellement ordonné sur base des résultats académiques des étudiants. Ainsi $\forall s_i, s_j \in S$, il existe une relation d'ordre tel que $s_i \geq s_j$ ou $s_j \geq s_i$.

De plus il faut considérer un ensemble fini T contenant les différents sujets proposés au étudiants.

Nous représenterons les choix de sujets réalisés par chaque étudiant par la fonction suivante : $\gamma : S \rightarrow T*$. Ainsi pour un étudiant s, $\gamma(s)$ représente les différents sujets $t_1, ..., t_j$ choisis par s, dans l'ordre de préférence de s (t_1 étant son premier choix, t_2 étant son second choix, ...). Notons que comme nous l'avons décrit plus haut $\gamma(s)$ ne contient qu'un sous ensemble de T. De plus, chaque étudiant n'aura pas nécessairement le même nombre de sujets dans sa liste de préférences.

Il est possible que certains sujets proposés admettent deux étudiants qui devront dès lors travailler en binôme sur le sujet. Ainsi nous expliciterons cette possibilité comme suit. Considérons $\gamma_s : S \times T \to S$ une fonction associant un second étudiant s a un couple (s,t), où s est un étudiant et $t \in \gamma(s)$ un sujet choisi par l'étudiant. Cette fonction (γ_s), est une fonction partielle, elle n'est pas nécessairement définie pour tou couple (s,t). De ce fait, pour un étudiant s et un sujet t nous écrirons $\gamma_s(s,t) = s$, si et seulement si, $(s,t) \in dom(\gamma_s)$ et γ_s associe s à (s,t).

Nous modéliserons une attribution par un "mapping" $M : S \to T$ associant un étudiant $s \in S$ à un sujet $t \in T$. Notons cependant qu'une attribution n'est pas nécessairement totald, il peut arriver que $dom(M) \subset S$, en d'autres mots que certains étudiants n'aient pas de sujet assigné. De plus M n'est pas obligatoirement injective, un sujet pouvant être assigné à un binôme d'étudiants).

Sur cette base, étant donné les ensemble S et T et les fonctions γ et γ_s, le problème est de trouver (si elle existe) une attribution M dite stable. Une attribution stable se définissant ainsi :

- tous les étudiants disposent d'un sujet : $dom(M) = S$
- chaque étudiant s'est vu attribuer un sujet faisant partie de sa liste de préférence : $\forall s \in dom(M) : M(s) \in \gamma(s)$
- si deux étudiants sont assignés à un même sujet, ils ont choisi de le faire ensemble : si $M(s) = M(s')$ alors soit $s = s'$ soit $\gamma_s(s, M(s)) = s'$ et $\gamma_{s'}(s', M(s')) = s$.
- l'attribution est stable que s'il n'existe pas de "meilleur" (sur base de $(S, \geq)$) étudiant qui aurait préféré t à son sujet courant : si $M(s_i) = t$ alors $\nexists j < i$ tel que $M(s_j) = t'$ et $\gamma(s_j) = \langle \ldots, t, \ldots, t', \ldots \rangle$

Sur cette base, il apparaît clairement que le problème d'assignation des mémoires est une instance du problème de mariage stable dans lequel les étudiants jouent le rôle des hommes tandis que les sujets celui des femmes ou inversément. Il existe cependant quelques différences entre ce problème et celui du mariage stable classique :

- Tout d'abord les listes de préférences des étudiants sont incomplètes et peuvent varier de taille selon les étudiants. On notera cependant que chaque étudiant fournit une liste de préférence complètement ordonnée (il n'existe pas d'égalité, pas de sujet à égalité).
- Par ailleurs, il est évident que tous les sujets disposent de la même liste de préférence. Cette liste est ordonnée comme décrit plus haut. Cependant, bien que cette liste de préférences soit complète, elle contient des égalités potenties. Les étudiants appartenant à S ne sont pas ordonnés sur base d'un ordre strict $(>)$ mais bien sur base d'un ordre permettant les égalités $(\geqslant)$.

3.2 L'algorithme

Avant d'étudier en détails l'algorithme, nous allons d'abord décrire son fonctionnement et les différentes notations utilisées.

La figure 3.1 illustre l'algorithme utilisé pour l'assignation des mémoires. L'algorithme utilise les éléments de S sous forme d'une liste $\langle s_1, \ldots, s_N \rangle$ tel que $s_i \geq s_{i+1}$ pour tout $1 \leq i \leq N$.

L'algorithme calcule un ensemble des attributions possibles. Il débute par une attribution initiale M_0 qui peut être vide tout comme déjà contenir une solution partielle qui aurait été fixée au préalable [1]

L'algorithme manipule une pile contenant des attributions partielles. Il va boucler en exploitant cette pile. A chaque itération, il prend l'attribution M se trouvant en tete de pile. Il va ensuite essayer d'étendre cette attribution partielle M en parcourant tous les étudiants s_i en ordre décroissant. Si l'étudiant s_i n'a pas (encore) d'assignation dans M, nous allons rechercher les sujets venant de sa liste de préférences n'ayant pas encore été assignés. De manière formelle, nous notons $\gamma(s_i)$ la liste de préférences d'un étudiant s_i. Ainsi la liste des sujets non encore assignés dans M sera notée $ff_M(\gamma(s_i))$. De plus, o_{f_1} représente le premier sujet libre restant et donc f_1 représente la position du sujet dans la liste de préférences d'origine. De ce fait, o_{f_1} est le $f_1{}^{\text{ème}}$ choix de l'étudiant, o_{f_2} est le $f_2{}^{\text{ème}}$ choix et ainsi de suite.

1. Ceci permet d'interagir avec l'algorithme en fixant une partie des assignations manuellement.

$T \leftarrow \emptyset$
$push(T, M_0)$
repeat
  $M \leftarrow pop(T)$
  for i *from* 1 *to* N *do*
    if $s_i \notin dom(M)$ *then begin*
      let $\langle o_{f_1}, \ldots, o_{f_k} \rangle = ff_M(\gamma(s_i))$
      if $k \geq 2$ *and* $f_2 \leq \Delta_c$ *then*
        if $\gamma_s(s_i, o_{f_2}) = s_j$ *and* $s_j \notin dom(M)$ *then*
          $push(T, M \cup \{(s_i, o_{f_2}), (s_j, o_{f_2})\})$
        else
          $push(T, M \cup \{(s_i, o_{f_2})\})$
      if $k \geq 1$ *and* $f_1 \leq \Delta_c$ *then*
        if $\gamma_s(s_i, o_{f_1}) = s_j$ *then*
          $M \leftarrow M \cup \{(s_i, o_{f_1}), (s_j, o_{f_1})\}$
        else
          $M \leftarrow M \cup \{(s_i, o_{f_1})\}$
      end
  output M
until $T = \emptyset$

FIGURE 3.1 – L'algorithme

L'algorithme se comportera de manière différente en fonction du nombre de sujets restants. S'il reste deux sujets, o_{f_2} existe, et que f_2 est inférieure ou égale au nombre maximum de choix que l'on souhaite prendre en compte (représenté par Δ_c) [2], on étendra M en attribuant le sujet o_{f_2} à l'étudiant s_i. Notons cependant qu'il est possible d'associer un second étudiant à un sujet si le sujet autorise deux étudiants et si la "place" est libre dans M. Cette nouvelle version de l'assignation est ensuite mise sur la pile pour un traitement ultérieur.
Ensuite si o_{f_1} existe, et que f_1 est inférieure ou égale au nombre maximum de choix que l'on souhaite prendre en compte (représenter par Δ_c), on étendra M en attribuant le sujet o_{f_1} à l'étudiant s_i (la possibilité d'avoir d'avoir deux étudiants associés au même sujet est prise en compte). On ne mettra cette fois pas l'assignation sur la pile car l'algorithme va continuer à s'exécuter en utilisant l'assignation étendue. L'algorithme continuera d'étendre M tout en mettant occasionnellement de nouveaux éléments sur la pile jusqu'à traiter tous les étudiants. Dans le cas où un étudiant s_k, n'a plus de sujet disponible, une assignation se construit mais s_k n'aura pas de sujet associé. Cela signifie que les attributions créées par l'algorithme ne sont pas nécessairement complètes.

2. Δ_c est un paramètre de l'algorithme. Il permet de définir le nombre de choix d'un étudiant que l'on souhaite prendre en compte.

Une fois tous les étudiants traités, l'attribution M produite est affichée/retournée, et les itérations continuent en prenant une attribution partielle se trouvant en tête de pile. L'algorithme continuera à s'exécuter jusqu'a ce que la pile soit vide.

3.3 extensions

Comme déjà décrit précédement l'algorithme permet de fixer préalablement certaines paires. Ceci peut impacter directement les résultats.

L'algorithme accepte actuellement 2 paramètres supplémentaires à prendre en compte. Tout d'abord le nombre maximum de sujets autorisés par promoteur (Δsub). Le second paramètre permet de définir le nombre maximum d'étudiant autorisé par promoteur ($\Delta stud$). Il relativement est simple d'exploiter ces paramètres au sein de l'algorithme. Une attribution M ne sera pas étendue que si par l'ajout d'une nouvelle paire, on venait à violer δsub ou $\delta stud$. Il peut être intéressant de s'attarder à cette extension afin de voir quels en sont les impacts.

Chapitre 4

Étude de l'algorithme d'assignation des mémoires

4.1 Introduction

Dans ce chapitre nous allons tenter de déterminer la compléxité de l'algorithme d'assignation des mémoires [16]. Pour y parvenir, nous allons préalablement répondre à plusieurs questions qui nous permettrons de catégoriser cet algorithme. Sur base de ces différentes observations nous pourrons alors déterminer sa complexité.

Pour plus de facilité dans notre étude, nous avons exclu la possibilité que deux étudiants travaillent sur le même mémoire, ce point ne modifiant en rien la complexité ou les caractéristiques premières de l'algorithme.

4.2 A quel forme de mariage stable avons-nous à faire ?

Avant toute chose, il est utile de s'intéresser à la forme de mariage stable associé au problème d'assignation de mémoire. Dans le chapitre précédent nous avons fait un rapide tour d'horizon de différentes formes de mariages stables existants. Nous allons maintenant tenter de rattacher l'algorithme d'assignation des mémoires à l'une de ces formes.

Nous allons, tout d'abord, instancier le concept de mariage stable dans ce cadre. Dans un mariage stable "classique", on considère un ensemble M qui contient l'ensemble des hommes ainsi qu'un ensemble W qui contient l'ensemble des femmes. Dans le cadre de ce document nous considérerons d'une part S comme l'ensemble des étudiants et d'autre part l'ensemble T contenant l'ensemble des mémoires. On notera M une association entre un étudiant et un mémoire tel que (s, t) où $s \in S$ et $t \in T$.

Afin d'assigner les mémoires, le processus nécessite que pour chaque $t \in T$ une liste finie de s tel que $s \in S$. Cette liste définit la préférence d'un sujet par

rapport aux différents étudiants sur base de leurs résultats académiques. Elle peut contenir des égalités. De plus, il est également nécessaire que pour chaque $s \in S$ une liste de $t \in T$ soit définie. Cette liste doit être finie et peut être incomplète [16]. Il est bien évident que les ensembles S et T peuvent être de taille différente.

Sur base de cette formulation, on peut considérer que l'on est face à une forme de mariage stable avec égalités et listes incomplètes (SMTI [1]).

Cependant, si l'on étudie plus en détail l'algorithme utilisé [16] on constate que l'on ne tient pas compte des égalités. En effet, la liste fournie est complètement ordonnée et ne contient aucune égalité. En réalité, avant l'exécution de l'algorithme, on va supprimer toute égalité de la liste. Pour ce faire, l'ordonnancement des éléments en égalité sur base d'un tri aléatoire va être forcé. L'algorithme traite en réalité des problèmes ayant la forme de mariage stable avec liste incomplète (SMI [2]) et plus précisément, de mariage stable avec listes incomplètes présentes d'un seul côté de l'assignation.

Une particularité de ce cas, est que la liste de préférences des étudiants établie sur base de leurs résultats académiques, pour chaque sujet, est identique. Chaque sujet dispose de la même liste de préférences pour les étudiants. Ce point fait l'objet d'une forme particulière de mariage stable appelé "Stable marriage with master list " (SM-ML) [14]. Dans le cas de l'assignation des mémoires, nous nous trouvons en fait dans un cas de SMI-1ML, soit un mariage stable avec liste incomplète et une "master" liste du coté des sujets.

4.3 Le concept d'attribution stable dans le cadre de l'assignation des mémoires

Afin de définir le concept d'attribution stable dans le contexte de l'assignation des mémoires, il est nécessaire de définir le concept de paire bloquante.
Nous trouvant dans le cas d'un SMI, nous nous référons à la définition de la paire bloquante vue au chapitre 2.

Il faut considérer un ensemble A contenant l'ensemble des paires acceptables pour une instance I d'un SMI donné. Nous noterons $|A|$ la taille de l'ensemble A. Dès lors une attribution M est un sous ensemble de A tel que, $|\{s : (s,t) \in M\}| \leq 1$ pour tout s et $|\{t : (s,t) \in M\}| \leq 1$ pour tout t. De ce fait, une paire $(s,t) \in A \backslash M$ bloque M si :

- s est non-associé ou préfère t à son "partenaire" dans M
- et t est non-associé ou préfère s à son "partenaire" dans M

Nous dirons donc qu'une attribution M est dite stable s'il n'existe pas de paire bloquante telle que définie plus haut.

1. Stable marriage with ties and incomplete list
2. stable marriage with incomplete list

Comme défini auparavant, nous ne sommes pas en présence d'une réelle forme de mariage stable avec égalité et liste incomplète (SMTI). L'algorithme se basant sur des listes sans égalité (SMI), il n'est donc pas cohérent de définir une forme de stabilité (faible stabilité, forte stabilité, super-stabilité).

4.4 Si une attribution stable existe, l'algorithme la trouve-t-il ?

On peut constater dès la première lecture qu'il est évident que l'algorithme construit des attributions et les affiche. Il reste cependant à vérifier si parmi les attributions générées, se trouve au moins une attribution stable et que l'algorithme la trouve.

Il est presque immédiatement prouvé que dans le cas le plus simple d'attribution stable, l'algorithme trouve bien le matching. Le cas le plus simple à considerer est le suivant : On considère un étudiant s_1 et un sujet t_1 ayant les preférences suivantes : $s_1 : t_1 \quad t_1 : s_1$ Il apparaît donc évident que la seule et unique paire possible est (s_1, t_1). Il est tout aussi évident que l'attribution qui sera créée est stable. En effet, il est impossible de trouver une paire bloquante telle que $(s,t) \notin M$, les seules valeurs acceptables pour s et t étant respectivement s_1, t_1.

Une fois les différentes variables initialisées et si l'on suit l'exécution de l'algorithme, il boucle tant que la pile n'est pas vide. Durant ces itérations, on retirera l'attribution en tête de pile et on travaillera ensuite avec cet élément. La première itération de la boucle de parcours des étudiants se résumera en une simple assignation vu que l'on ne considère qu'un seul étudiant. Etant donné cette première itération et le fait qu'aucune attribution n'a encore été réalisée, l'étudiant s_1 n'aie pas de sujet assigné, la première condition sera consécutivement vrai. La liste des sujets encore disponibles pour s_1 contiendra le seul et unique sujet présent dans sa liste de préférences (t_1) puisqu'aucun sujet n'a encore été assigné à ce stade. La condition vérifiant que le nombre de sujets encore disponibles est supérieur à 2 sera fausse et le code suivant ne sera donc pas exécuté. Par contre la seconde condition sera belle et bien vraie vu qu'on dispose d'un sujet disponible et qu'on présume Δ_c supérieur ou égal à 1. Dans un soucis de facilité on considèrera que t_1 n'accepte qu'un seul étudiant, la condition vérifiant ce point sera donc fausse et le code associé ne sera donc pas exécuté. L'algorithme passera donc au "else" où la paire (s_1, t_1) sera ajoutée à l'attribution M. N'ayant plus d'étudiant la boucle prendra fin pour parvenir à l'affichage de l'attribution calculée. L'algorithme prendra ensuite fin de part le fait que la pile sera vide à ce stade.

Nous venons de voir que l'algorithme a produit une attribution $\langle (s_1, t_1) \rangle$ stable.

Sur cette première base, il est quasi immédiat de prouver que l'algorithme sera toujours en mesure de trouver une attribution si elle existe. En effet, de part le fonctionnement de l'algorithme, les mémoires seront toujours assignés sur

base de la liste des étudiants, cette liste étant la master list correspondant aux préférences des sujets. La première attribution créée contiendra des associations où chaque étudiant se verra attribuer le premier sujet encore disponible dans sa liste de préférences. De ce fait la première attribution créée sera forcément stable. Pour le prouver tentons de trouver une paire bloquante telle que définie plus haut (cfr 4.3). Pour créer une telle paire il faudrait assigner un autre sujet à un étudiant ou un autre étudiant à un sujet. Considérons une paire $(s,t) \notin M$ comme proposition de paire bloquante. Sachant que pour l'attribution M, nous avons les paires (s,t') et (s',t). De part le fonctionnement intrinsèque de l'algorithme, cela signifierait donc soit attribuer un "moins bon" étudiant à t, soit attribuer "un moins bon sujet" à s ce qui ne répond pas à la définition de paire bloquante. Il est donc évident que dans ce cadre l'algorithme est en mesure de trouver une attribution stable si elle existe.

On ne peut garantir l'existence d'une attribution stable si le mécanisme donnant la possibilité de paramétrer l'algorithme en fournissant une série de paires prédéfinies est utilisé. Cependant, de part son fonctionnement, l'algorithme sera toujours en mesure de trouver une paire stable si elle existe. Le fait de fixer une série de paires ne fait que limiter les possibiltés d'attribution.

4.5 Si plusieurs attributions stables existent, l'algorithme les trouve-t'il toutes ?

Nous venons de prouver que l'algorithme est en mesure de trouver une attribution stable s'il en existe une. Nous allons maintenant nous intéresser à la capacité de l'algorithme à trouver tous les matching existants.

A l'étude attentive de l'algorithme, on se rend compte que pour chaque étudiant un nombre limité de sujets est pris en compte. A chaque itération de l'algorithme, on étudiera les possibilités pour les 2 premiers sujets restants. De part les permutations réalisées par l'algorithme, on peut cependant générer un nombre important de combinaisons et d'associations.

Pour être exact, l'algorithme génère en fait au maximum 2^n associations où n est égale à $|S|$. Ce nombre est tout à fait cohérent par rapport à l'algorithme vu que celui-ci effectue en réalité 2 choix $|S|$ fois ; ceci correspond à une permutation mathématique classique. Théoriquement une permutation de ce type est une permutation avec répétitions. Il semble étonnant que l'on soit face à ce type de permutation car conceptuellement un même choix ne peut être répéter. Ceci est par ailleurs garanti par les différentes conditions de création des paires.

Afin de garantir l'obtention d'un tel nombre d'associations et de permettre la génération de toutes les solutions, il est nécessaire que les listes de préférences des étudiants n'entrent pas en conflit entre elles ou soient d'une taille de 2 et ce malgré les conflits entre les listes obtenues après suppression des sujets déjà sélectionnés. L'ensemble des choix étant différent à chaque itération, les permutations sensées produire des répétitions ne se produisent pas et il ne peut donc y avoir répétition de choix.

L'ensemble de choix étant limité aux 2 premiers éléments de la liste, on pourrait penser que des attributions stables soient ignorées car elle nécessite de choisir le $3^{ème}$ élément de cette liste lors d'une itération.

Cependant, bien que l'on puisse clairement ignorer certaines attributions de part le fonctionnement de l'algorithme, l'algorithme est toujours en mesure de trouver toutes les attributions. Au premier abord cette affirmation peut paraitre étrange, mais elle peut être aisément justifiée.

Comme nous l'avons déjà décrit dans le point précédement 4.4, l'algorithme se base tout d'abord sur la master liste des sujets, c'est à dire qu'il parcoure les étudiants sur base de leur performance académique. L'algorithme peut ensuite créer une nouvelle attribution en prenant le second choix de l'étudiant, mais on ajoutera également une association entre l'étudiant et son premier choix parmi les sujets restants, s'il en reste. La première attribution qui sera créée, contiendra en fait tout les premiers choix des étudiants. Comme nous l'avons déjà dit cette attribution est stable, elle est en faite la seule attribution stable qui sera générée. Toutes les attributions qui seront créée par la suite résultent d'une ou plusieurs associations entre des étudiants et leur second choix parmi les sujets restant. De ce fait, et vu qu'on itère sur base de la master liste, chacune des paires ainsi créées associe soit un "moins bon" étudiant à un sujet soit un "moins bon" sujet à un étudiant. De ce fait il est évident que l'on peut trouver des paires bloquantes pour ces attributions.

On constate de ce fait qu'il n'existe qu'une et une seule attribution stable correspondant à la définition donnée d'attribution stable

4.6 Y-a-t-il une relation d'ordre dans la découverte des différentes attributions ?

Afin d'aller plus en avant dans les détails de l'algorithme, nous allons maintenant essayer d'identifier un possible ordonnancement dans la découverte des différentes attributions.

4.6.1 Stabilité maximum

Le premier point que nous allons étudier, est la possibilité que les attributions trouvées soit ordonnées selon leur stabilité. Autrement dit, que les attributions trouvées soient ordonnées par le nombre de paires bloquantes potentielles que l'on peut leur associer.

Nous savons déjà que la première attribution trouvée est une attribution stable, c'est à dire qu'elle n'existe aucune paire bloquante. Voyons maintenant si les autres éléments respectent cet ordre.

Il suffit d'un court exemple pour démontrer qu'il n'existe pas de relation d'ordre sur le critère du nombre de paires bloquantes associées.

Considérons un ensemble S tel que $(s_1, s_2, s_3, s_4) \in S$ et un ensemble T tel que $(t_1, t_2, t_3, t_4) \in T$. La "master" liste pour T étant la suivante (s_1, s_2, s_3, s_4).

Les préférences pour les éléments de S étant celles-ci :

$$\begin{aligned} s_1 &: \quad t_1, t_2, t_3, t_4 \\ s_2 &: \quad t_1, t_2, t_3, t_4 \\ s_3 &: \quad t_1, t_2, t_3, t_4 \\ s_4 &: \quad t_1, t_2, t_3, t_4 \end{aligned}$$

Une fois l'algorithme exécuté on obtient les attributions suivantes dans cet ordre précis :

associations	nombre de paires bloquantes
$\{(s_1,t_1),(s_2,t_2),(s_3,t_3),(s_4,t_4)\}$	0
$\{(s_1,t_1),(s_2,t_2),(s_3,t_4),(s_4,t_3)\}$	1
$\{(s_1,t_1),(s_2,t_3),(s_3,t_2),(s_4,t_4)\}$	1
$\{(s_1,t_1),(s_2,t_3),(s_3,t_4),(s_4,t_2)\}$	2
$\{(s_1,t_2),(s_2,t_1),(s_3,t_3),(s_4,t_4)\}$	1
$\{(s_1,t_2),(s_2,t_1),(s_3,t_4),(s_4,t_3)\}$	2
$\{(s_1,t_2),(s_2,t_3),(s_3,t_1),(s_4,t_4)\}$	2
$\{(s_1,t_2),(s_2,t_3),(s_3,t_4),(s_4,t_1)\}$	3

Il apparaît clairement de cet exemple que l'algorithme ne produit pas les attributions dans un ordre lié au nombre de paires bloquantes associées.

4.6.2 Nombre de paires

Une autre possibilité envisageable est que l'algorithme propose en priorité les solutions minimisant le nombre de membres célibataires. Il proposerait les attributions par leur niveau de complétude[3].

Cependant, il suffit à nouveau d'un simple exemple pour rapidement écarter cette hypothèse.

Considérons un ensemble S tel que $(s_1, s_2, s_3, s_4) \in S$ et un ensemble T tel que $(t_1, t_2, t_3, t_4) \in T$. La "master" liste pour T étant la suivante (s_1, s_2, s_3, s_4). Les préférences pour les éléments de S étant :

$$\begin{aligned} s_1 &: \quad t_1, t_4, t_3 \\ s_2 &: \quad t_4, t_2, t_3 \\ s_3 &: \quad t_1, t_4, t_3 \\ s_4 &: \quad t_2, t_4, t_1 \end{aligned}$$

3. On dit qu'une attribution est complète lorsqu'elle ne laisse aucun membre célibataire. Dans le cas de l'attribution des mémoires cette notion est dérivée de la manière suivante : on considère une attribution complète lorsque chaque étudiant dispose d'un sujet.

L'algorithme fournira dans ce cas les associations suivantes dans cet ordre précis :

associations	nombre d'étudiants sans sujet
$\{(s_1,t_1),(s_2,t_4),(s_3,t_3),(s_4,t_2)\}$	0
$\{(s_1,t_1),(s_2,t_2),(s_3,t_4)\}$	1
$\{(s_1,t_1),(s_2,t_2,(s_3,t_3),(s_4,t_4))\}$	0
$\{(s_1,t_4),(s_2,t_2,(s_3,t_1)\}$	1
$\{(s_1,t_4),(s_2,t_2,(s_3,t_3),(s_4,t_1))\}$	0
$\{(s_1,t_4),(s_2,t_3,(s_3,t_1),(s_4,t_2))\}$	0

Comme illustré dans le tableau ci-avant, on constate que les résultats obtenus ne sont pas ordonnés sur base de la complétude des associations.

4.6.3 "Master" liste

De part le fait que l'algorithme itère sur base de la "master" liste contenant les étudiants, il est possible qu'une relation d'ordre apparaisse entre celle-ci et les attributions produites.

Si on étudie le fonctionnement de l'algorithme, on peut constater que pour créer les différentes paires on se base essentiellement sur la "master" liste. Nous avons déjà constater que l'algorithme génère une d'attributions

Cette génération se fait dans l'ordre suivant.

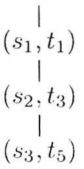
- On commence par essayer de construire des paires en prenant le premier choix restant dans la liste de préférences.
- On envisage, ensuite, les différentes alternatives possibles sur base de cette liste en attribuant le second choix possible.

Cette attribution de second choix se fait en ordre inversé par rapport à la liste de préférence de T. Cette manière de créer des paires est assez similaire au processus de "backtracking"[9],[1] que l'on retrouve dans des langages tels que Prolog. Ce processus de "backtracking" va faire en sorte qu'on va réévaluer les choix qu'on a réalisés dans l'ordre inverse duquel on les a fait.

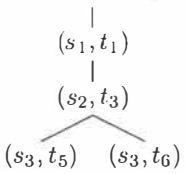
Afin d'illustrer ce phénomène, considérons l'exemple suivant. Soit un ensemble S tel que $(s_1,s_2,s_3) \in S$ et un ensemble T tel que $(t_1,t_2,t_3,t_4,t_5,t_6) \in T$. La "master" liste pour T étant la suivante (s_1,s_2,s_3). Les préférences pour les éléments de S étant celles-ci :

$$\begin{array}{ll} s_1 : & t_1,t_2 \\ s_2 : & t_3,t_4 \\ s_3 : & t_5,t_6 \end{array}$$

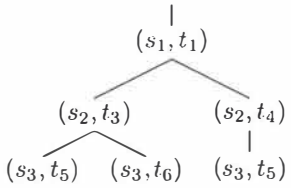
La première attribution générée sera la suivante : $\{(s_1,t_1),(s_2,t_3,(s_3,t_5)\}$
Ce qui correspond à l'arbre suivant :



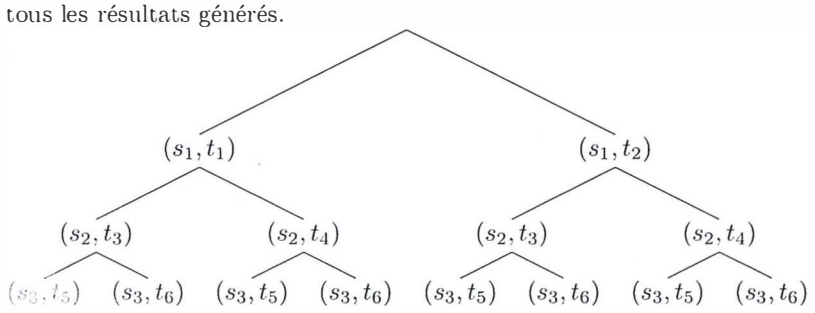
L'itération suivante produira l'attribution :$\{(s_1, t_1), (s_2, t_3, (s_3, t_6)\}$
L'arbre évalué aura donc évolué tel que :



L'itération suivante produira l'attribution : $\{(s_1, t_1), (s_2, t_4, (s_3, t_5)\}$
L'arbre évalué aura donc évolué vers ceci :



L'algorithme va continuer à itérer jusqu'à obtenir l'arbre suivant qui couvre tous les résultats générés.



Une bonne manière de comprendre l'ordre dans lequel les différentes attributions sont générées est d'associer un poids à chaque paire calculée. On attribue au premier choix une valeur plus importante qu'aux autres.

Pour illustrer ce point, repartons de l'exemple précédent. Si on attribue à la paire (s_i, t_j) tel que $t_j = o_{f_1}$ un poids correspondant à 2^r où r représente $|S|$ moins la position de s_i dans la "master" liste de T. Nous utiliserons cette formule car elle permet d'attribuer un poids différent à chaque paire. Si on compare le poids total de toutes les associations générées on obtient :

associations	poids total
$\{(s_1,t_1),(s_2,t_3),(s_3,t_5)\}$	7
$\{(s_1,t_1),(s_2,t_3),(s_3,t_6)\}$	6
$\{(s_1,t_1),(s_2,t_4),(s_3,t_5)\}$	5
$\{(s_1,t_1),(s_2,t_4),(s_3,t_6)\}$	4
$\{(s_1,t_2),(s_2,t_3),(s_3,t_5)\}$	3
$\{(s_1,t_2),(s_2,t_3),(s_3,t_6)\}$	2
$\{(s_1,t_2),(s_2,t_4),(s_3,t_5)\}$	1
$\{(s_1,t_2),(s_2,t_4),(s_3,t_6)\}$	0

Afin de facilité la compréhension, vous trouverez le détails du calcul du poids pour les deux premières attributions.

Pour $\{(s_1,t_1),(s_2,t_3),(s_3,t_5)\}$ on considère : $2^{3\text{-}1} + 2^{3\text{-}2} + 2^{3\text{-}3} = 7$

Pour $\{(s_1,t_1),(s_2,t_3),(s_3,t_6)\}$ on considère : $2^{3\text{-}1} + 2^{3\text{-}2} = 6$

L'ordre apparaît immédiatement. Au départ d'associations où l'on maximise le nombre de premier choix, on se dirige vers les associations où l'on minimise le nombre de premier choix, et ce toujours dans les premiers éléments de la "master" liste de T.

4.7 Temps au pire pour trouver la première paire stable

Un point d'intéret particulier dans l'étude de cette algorithme est de connaitre le temps au pire pour trouver la première paire stable. De part le fonctionnement intrinsèque de l'algorithme, trouver la première paire stable est presque immédiat.

Considérons la situation suivante :

- S tel que $(s_1,...,s_n)$, n étant égale à $|S|$
- T tel que $(t_1,...,t_m)$, m étant égale à $|T|$
- ml comme la liste de préférences des étudiants pour T
- Chaque $s_i \in S$ a définit une liste de préférence $\gamma(s_i)$.

Exécutons maintenant l'algorithme. Après avoir initialisé les différentes variables, entamons la première boucle. L'algorithme prend le premier élément de la pile et commence à itérer sur les étudiants. Il sélectionne donc un étudiant s_i. L'ensemble de matching venant d'être initialisé, s_i n'a pas encore de paire dans M. L'algorithme va donc tenter de créer une paire. Il va donc rechercher les préférences de s_i, $ff_M(\gamma(s_i)$. S'agissant de la première itération, $ff_M(\gamma(s_i)$ sera égale à $\gamma(s_i)$.
Si cette liste contient au moins 2 éléments, il créera un nouveau matching qu'il étendra avec la paire (s_i, o_{f_2}) et l'ajoutera à la pile. Si la liste contient ensuite au moins 1 élément, il étendra le matching courant en y ajoutant la paire (s_i, o_{f_1}). De part le fait que $(s_i, o_{f_1}) = ff_M(\gamma(s_i)$, o_{f_1} est en fait le premier choix de s_i, il suffit à dire que la paire ainsi créé est optimale.

De ce fait, on peut dire que le temps au pire pour calculer la première attribution stable est de l'ordre de $O(|S|)$ et ce à condition que les opérations de calcul de préférence et de construction d'attribution se font en un temps $O(1)$. Ceci correspond uniquement au cas où aucun matching n'a été prédéfini auparavant comme l'algorithme le permet [4].

Si une paire devait être définie au préalable, il serait alors peut être nécessaire de parcourir tout l'arbre des possibilités avant de parvenir à la conclusion qu'il n'existe pas d'attribution stable. Dans le cas contraire la première attribution stable sera trouvée dans le même temps que dans le cas classique.

4.8 Temps au pire pour trouver les différentes attributions

Comme nous l'avons vu en 4.5, de part la structure du problème il n'existe qu'une attribution stable. Celle-ci sera trouvée en un temps $O(n)$, n étant égal à $|S|$. De l'ensemble des attributions découvertes, l'algorithme va cependant en découvrir quelques unes plus intéressantes. Il semble donc intéressant de connaitre le temps au pire afin de calculer ces différentes attribution.

Pour les calculer il est nécessaire de parcourir l'ensemble de l'arbre des possibilités. Le temps au pire pour calculer cet ensemble est donc de l'ordre de $O(2^n)$, n étant égal à $|S|$ tel que vu précédemment en 4.6.3.

4.9 Remarques

Comme nous l'avons vu au travers de ce chapitre, l'algorithme d'assignation des mémoires ignore un nombre important d'attributions possibles. Bien que ces attributions ne soit pas stable par définition, certaines d'entre elles pourraient être intéressantes.

Nous sommes concentré au travers de ce documents sur la notion d'attribution stable. Il serait intéressant d'approfondir à la notion de regret minimum dans le cadre de l'attribution des mémoires. On définit la notion de regret comme suit : Pour une instance I de SM, considérons une attribution M. Pour une paire $(m, w) \in M$, le "regret" de m sera égale à la position de w dans la liste de préférences de m. Le "regret de w sera, logiquement, égale à la position de m dans la liste de préférences de w [3].

L'application de la notion de regret minimum sur un problème de mariage stable revient à tenter de trouver une attribution où la satisfaction moyenne des participants est plus élevé. En effet, l'attribution minimisera le "regret" de chacun des participants.

Exemple :

4. voir chapitre 3.2

Considérons l'ensemble M des hommes et leurs préférences :
- Tom : ⟨ *Emma, Kristen, Nina* ⟩
- John :⟨ *Emma, Nina, Kristen* ⟩
- Alex :⟨ *Nina, Emma, Kristen* ⟩

Considérons l'ensemble W des femmes et leurs préférences :
- Nina : ⟨ *Tom, John, Alex* ⟩
- Kristen : ⟨ *Tom, John, Alex* ⟩
- Emma : ⟨ *Tom, John, Alex* ⟩

Pour l'attribution suivante le regret total sera de 6. ⟨ { *Tom, Emma* }, { *John, Nina* }, { *Alex, Kristen* } ⟩

Tandis que pour cette nouvelle attribution le regret total sera de 4. ⟨ { *Tom, Kristen* }, { *John, Emma* }, { *Alex, Nina* }⟩

Dans le cas de l'attribution des mémoires, il se pourrait que certaines attributions non calculées aient un regret total inférieur à celle qui sont générées.

Chapitre 5

Conclusion

Au travers de ce document nous avons essayé d'étudier l'algorithme utilisé aux Facultés Notre Dame de la Paix de Namur dans le cadre de l'assignation des mémoires. Cet algorithme est basé sur le problème bien connu du mariage stable.

Dans ce cadre, nous avons tout d'abord fait un rapide état de l'art dans le domaine. Nous avons présenté les formes principales de mariage stable et leurs caractéristiques et avons pu ainsi présenter la forme classique de mariage stable. Nous avons également étudié des formes avec égalités et listes incomplètes. Ce parcours nous a également permis d'appréhender le concept d'attribution stable et de ses variations en fonctions du contexte.

Nous sommes ensuite entrés dans le vif du sujet en présentant l'algorithme utilisé. Nous avons présenté l'algorithme dans sa version formelle et avons décrit son fonctionnement.

Dans une troisième partie, nous avons abordé l'étude de l'algorithme au travers de divers points clés. Nous avons tout d'abord catégorisé l'algorithme. Pour ce faire, nous avons déterminé à quelle forme de mariage stable nous avions à faire. Par la même occasion, certaines spécificités de l'algorithme ont pu être mises en avant.

Cette première catégorisation étant réalisée, il s'est avéré nécessaire de définir le concept de stabilité dans le contexte de l'assignation des mémoires. Ainsi, sur base des points découverts au préalable, nous avons pu adapter la définition d'attribution stable à notre contexte.

Nous avons ensuite voulu vérifier si l'algorithme réalisait bien le travail attendu. Nous avons démontré que l'algorithme était en mesure de trouver une attribution stable si elle existe. Nous avons découvert que le fait de fixer une paire pouvait engendrer quelques soucis.

Conceptuellement, nous avons démontré qu'il ne pouvait exister qu'une seule attribution stable. Cette attribution stable sera toujours découverte car l'algorithme est définit pour découvrir une attribution stable si elle existe.

L'étape suivante de notre étude à porté sur l'ordre des solutions obtenues. Il semblait intéressant de voir s'il existait une relation d'ordre entre les différentes associations générées. Pour ce faire, nous avons tout d'abord étudié si l'ordre était lié à la stabilité des attributions. Cette hypothèse s'est rapidement avérée fausse.

Nous avons dès lors évalué une autre piste : laquelle consistait à vérifier une relation d'ordre basée sur la complétude des attributions. Les attributions générées aurait été ordonnées sur base du nombre de paires contenues. Cette hypothèse s'est également révélé fausse.

Nous avons, dès lors, étudié la possibilité que l'ordre de préférence pour les étudiants puisse déterminer une relation d'ordre. Cette proposition s'est avérée exacte puisque l'algorithme favorise les premiers choix pour les premiers éléments de la liste de préférences dans les premières associations.

L'étape suivante de notre étude fut la détermination du temps au pire afin de trouver la première attribution ainsi que toutes les attributions que l'algorithme est en mesure de trouver. Nous avons ainsi pu déterminer que trouver la première attribution stable est un travail relativement simple si l'on considère le fonctionnement de base de l'algorithme.

Cependant, ce travail peut devenir plus complexe si l'on prédéfinit certaines paires préalablement comme l'algorithme l'autorise.

Dans le cadre de la recherche de toutes les attributions, nous avons pu rapidement définir le temps au pire en nous basant sur les différentes observations déjà réalisées. Sur cette base, nous avons déterminé que nous étions face à un problème exponentiel.

Enfin, nous avons émis quelques remarques sur base des observations précédentes. Nous avons ainsi suggéré une étude du concept d'attribution avec regret minimum dans le cadre de l'algorithme d'attribution des mémoires

Au travers de ce document nous avons pu découvrir plusieurs facettes de l'algorithme. Il en reste, sans doute, de nouvelles à découvrir. Il serait très certainement opportun de s'intéresser aux optimisations envisageables. Celles-ci pouvant se faire au travers de modifications ponctuelles afin d'optimaliser le traitement et/ou l'amélioration de l'obtention des résultats mais également, en envisageant de nouvelles techniques de calcul.

Bibliographie

[1] Backtracking. http ://en.wikipedia.org/wiki/Backtracking, Dernière modification 27/06/2013, consulté le 12/07/2013.

[2] D. Gale and S. J. Shapley. College admission and the stability of marriage. jan 1962.

[3] Dan Gusfield. Three fast algorithms for four problems in stable marriage. jul 1985.

[4] Dan Gusfield and Robert W. Irving. Stable marriage and indifference. MIT press.

[5] Robert W. Irving. An efficient algorithm for the "stable roomates" problem. In *Journal of Algorithms 6*, pages 577 – 595. 1984.

[6] Robert W. Irving. Stable marriage and indifference. feb 1990.

[7] R.W. Irving. Stable marriage and indifference. Discrete Applied Mathematics, 1994.

[8] Kazuo Iwama and Shuichi Miyazaki. A survey of the stable marriage problem and its variants. In *International Conference on Informatics Education and Research for Knowledge-Circulating Society.*

[9] Jean-Marie Jacquet. *Programmation déclarative*, chapter Prolog. 2006.

[10] David F. Manlove.

[11] David F. Manlove. The structure of stable marriage with indifference. 2002.

[12] D. McVitie and L.B. Wilson. table marriage assignment for unequal sets. Kluwer Academic Publishers, 1970.

[13] DavidF. Manlovea Robert W. Irving Kazuo Iwamab Shuichi Miyazakib Yasufumi Moritab. Hardvariants of stable marriage. page 261 â 279. Theoretical Computer Science, 2002.

[14] David F. Manlove Robert W. Irving and Sandy Scott. The stable marriage problem with master preference lists. jul 2006.

[15] Sandy Scott. *A Study Of Stable Marriage Problems With Ties.* PhD thesis, University of Glasgow, jan 2005.

[16] Wim Vanhoof. *Stable Marriage for Thesis Assignment.* Facultés Notre-Dame de la Paix, Namur, mar 2013.

Annexe A

"Instanciation" de l'algorithme

Vous trouverez en annexe un projet Java contenant une instanciation de l'algorithme d'assignation des mémoires. Cette solution fait abstraction de l'assignation de plusieurs étudiants à un même sujet. Cette implémentation a permis une meilleure compréhension de l'algorithme. De part cette implémentation, une seconde validation des différentes propositions a pu être réalisée.

Annexe B

Description de l'algorithme

Ce document écrit par monsieur Wim Vanhoof, contient la description du problème d'assignation de mémoire ainsi que la description de l'algorithme utilisé. Il a servi de base au différentes recherches réalisées au travers de ce mémoire.