Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche

researchportal.unamur.be

University of Namur

THESIS / THÈSE

MASTER EN SCIENCES MATHÉMATIQUES

VirtualBelgium : modélisation et simulation des mariages et divorces

PICARD, Gaëlle

Award date:
2014

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 23. Jun. 2020

UNIVERSITE DE NAMUR

Faculté des Sciences

VirtualBelgium : Modélisation et Simulation des mariages et divorces

Mémoire présenté pour l'obtention

du grade académique de master en « Master en Sciences Mathématiques à finalité spécialisée »

Gaëlle PICARD

Août 2014

Université de Namur

VirtualBelgium :
Modélisation et Simulation des mariages et divorces

Gaëlle Picard
Promoteur : Philippe Toint
Copromoteurs : Éric Cornélis et Renaud Lambiotte

Année académique 2013-2014

Ce travail a bénéficié des ressources de la plate-forme technologique en calcul intensif (PTCI) de l'Université de Namur (FUNDP), Belgique, financées grâce au soutien du F.R.S.-FNRS (conventions No. 2.4617.07 et 2.5020.11) et des FUNDP ainsi que des ressources du Consortium des Équipements de Calcul Intensif (CECI).

Abstract

This master's thesis presents and analyses several marriages and divorces models. Diverse mate-search heuristics, mutual or not, are compared according to their pros and cons but also according to their adaptability to the problem. All models are then added to VirtualBelgium, a multiagent, population's evolution platform that is based on a validated synthetic population for the country. The installation process and the libraries used are detailed with care. The code that has been written in the context of this master's thesis is explained step by step through every function created. Moreover, all difficulties encountered are listed and solutions or thoughts to get round them are proposed. Finally, the theoritical analysis is completed with an empirical one with the comparison of our different models with reality with special care towards the explanation of the gap between simulation and reality.

Keywords : Wedding, divorce, stable marriages, mate-search, mutual search, multiagent systems

Résumé

Dans ce mémoire, nous présentons et analysons plusieurs modèles de mariages et divorces. Différentes heuristiques de recherche d'un partenaire sont comparées selon leur différentes qualités et leur adaptation au problème, que la recherche soit unilatérale ou bilatérale. Les modèles sont ensuite implémentés dans VirtualBelgium, une plateforme multi-agents de simulation d'évolution de population basée sur une population synthétique initiale validée et à taille réelle. Le fonctionnement de cette plateforme, son installation et les librairies utilisées sont détaillés. L'implémentation des différents modèles est expliquée via un détail de chaque méthode créée. De plus, les difficultés et restrictions rencontrées sont présentées et des solutions ou des pistes sont suggérées afin de les contourner. Enfin, l'analyse théorique des modèles est complétée par une analyse des résultats empiriques obtenus grâce à l'exécution du programme. Ces résultats sont comparés aux statistiques réelles de la Belgique avec un soin particulier apporté à l'explication des différences obtenues.

Mots-clés : Mariages, divorces, mariages stables, recherche d'un conjoint, recherche mutuelle, systèmes multi-agents

Remerciements

Je tiens à exprimer toute ma reconnaissance à mon promoteur Philippe Toint. Je le remercie de son encadremen, de ses précieux commentaires et ses remarques pertinentes.

J'adresse également mes sincères remerciements à mes deux co-promoteurs Renaud Lambiotte et Éric Cornélis pour leur disponibilité et leurs conseils.

Un remerciement particulier est adressé à Benjamin pour m'avoir soutenue et supportée même dans les moments les plus difficiles. Merci pour ton écoute, tes conseils et ton soutien. Merci tout simplement pour ta présence à mes côtés qui a rendu ces derniers mois plus agréables.

Bien évidemment, je me dois de remercier ma famille pour leur soutien inconditionnel. Merci Bobonne, Maman et Papa pour la confiance que vous m'avez toujours accordée. Je remercie également mes frères et soeurs pour les soirées de détente en famille.

Vient le moment de remercier tous mes amis sur qui j'ai pu compter. Merci particulièrement aux personnes qui m'ont soutenue ces deux derniers mois, merci pour le soutien et les parties de belote parfois bien nécessaires.

Du fond du coeur, merci à tous.

Table des matières

Introduction **1**

1 Présentation de VirtualBelgium **3**
1.1 Génération d'une population synthétique 3
1.2 Les logiciels utilisés 5
1.2.1 Repast HPC 5
1.2.2 MATSim 6
1.3 Cœur du programme 6
1.3.1 Classes générales 7
1.3.2 Individus et Ménages 8
1.3.3 Réseau 9
1.4 Programmation parallèle 10
1.5 Conclusion 11

2 Introduction aux systèmes multiagents **13**

3 Première modélisation des mariages et divorces **17**
3.1 Introduction aux méthodes de choix discrets 17
3.2 Choix entre mariage et célibat 19
3.3 Choix du partenaire : seul ou avec enfants 22

4 Problème des mariages stables **25**
4.1 Introduction 25
4.2 Problème classique 25
4.3 Quelques extensions au problème 31
4.3.1 Ensembles de tailles différentes 31
4.3.2 Listes incomplètes 32
4.3.3 Indifférences 33

4.3.4 Un peu plus d'équité 33
4.4 Caractéristiques recherchées chez un partenaire 36

5 Un deuxième modèle pour les mariages : The Wedding Ring 39

6 Analyse d'heuristiques de recherche séquentielle 45
6.1 Introduction 45
6.2 Trois approches du problème 46
6.3 Recherche unilatérale 48
6.4 Recherche bilatérale 52

7 Principes du logiciel Repast HPC 61
7.1 Simulation parallèle 62
7.2 Écrire un modèle Repast HPC 63
7.2.1 Classes Agent et code Package 63
7.2.2 La classe « Model » 65
7.2.3 La fonction `main` 66

8 Implémentation 67
8.1 Introduction 67
8.2 Méthodes communes aux trois modèles 67
8.2.1 Méthode simulant un mariage 67
8.2.2 Méthode simulant un divorce 68
8.2.3 Définition du potentiel attractif des individus 68
8.3 Description des modèles 69
8.3.1 Premier modèle : mariages et divorces 69
8.3.2 Deuxième modèle : mariages 71
8.3.3 Troisième modèle : mariages 73
8.3.4 Quatrième modèle : divorces 74
8.4 Résultats et comparaison des modèles 74
8.4.1 Résultats pour les mariages 74
8.5 Résultats pour les divorces 82

Conclusion et perspectives 85

A Principes de programmation orientée objet et bases de C++ 87
A.1 Classes et instances 87
A.2 Héritage 91
A.3 Polymorphisme et classes abstraites 92

B Codes pour les simulations : recherche unilatérale **95**
B.1 Code principal 95
B.2 Fonctions utilisées 97

C Codes pour les simulations : recherche bilatérale **99**
C.1 Code principal 99
C.2 Fonctions utilisées 103

D Installation de VirtualBelgium **111**
D.1 Installation 111
D.2 Téléchargements 111
D.3 Repast HPC 113
D.4 Compilation et exécution de VirtualBelgium 114
D.5 Configuration de VirtualBelgium 115
D.6 Mise à jour du code 116

Table des figures

1.1 Structure de VirtualBelgium 4

3.1 Graphe de f_{age} en fonction de l'âge 20
3.2 Graphe de f_{stay} en fonction de age_{change} 21
3.3 Graphes de Δ_e et Δ_n en fonction de l'âge 23
3.4 Graphes des utilités V_e et V_n en fonction de l'âge selon le nombre d'enfants n 24

5.1 Social network in the Wedding Ring – Tirée de [7] 40
5.2 Fonction de la pression sociale sp en fonction du nombre d'agents mariés dans le réseau des personnes significatives d'un individu pom – Tirée de [7] – $\beta = 7, \alpha : 0.5$ 41
5.3 Fonction de l'influence de l'age ai – Tirée de [7] 41

6.1 Chance de trouver un partenaire d'une certaine tranche de la population en fonction du nombre de candidats évalués avant la détermination d'un seuil d'exigence - $N = 100$ - Distribution uniforme 50
6.2 Chance de trouver un partenaire d'une certaine tranche de la population en fonction du nombre de candidats évalués avant la détermination d'un seuil d'exigence - $N = 1000$ - Distribution uniforme 51
6.3 Chance de trouver un partenaire d'une certaine tranche de la population en fonction du nombre de candidats évalués avant la détermination d'un seuil d'exigence - Comparaison $N = 100$ et $N = 1000$ - Distribution uniforme 52
6.4 Nombre de couples formés lors d'une recherche séquentielle bilatérale et selon la méthode utilisée - Distribution uniforme . 56

6.5 Valeur moyenne des individus ayant trouvé un partenaire lors d'une recherche séquentielle bilatérale et selon la méthode utilisée - Distribution uniforme . . . 57
6.7 Nombre de couples formés lors d'une recherche séquentielle bilatérale et selon la méthode utilisée - Distribution normale N(50,10) . . . 57
6.6 Différence moyenne de valeur au sein des couples formés lors d'une recherche séquentielle bilatérale et selon la méthode utilisée - Distribution uniforme . . . 58
6.8 Valeur moyenne des individus ayant trouvé un partenaire lors d'une recherche séquentielle bilatérale et selon la méthode utilisée - Distribution normale N(50,10) . . . 58
6.9 Différence moyenne de valeur au sein des couples formés lors d'une recherche séquentielle bilatérale et selon la méthode utilisée - Distribution normale N(50,10) . . . 59

8.1 Comparaison du nombre de mariages selon l'âge des époux - Modèle 1 . . . 75
8.2 Comparaison du nombre de mariages selon l'âge des épouses - Modèle 1 . . . 75
8.3 Comparaison du nombre de mariages selon l'âge des époux - Modèle 2 . . . 76
8.4 Comparaison du nombre de mariages selon l'âge des époux - Modèle 3 . . . 76
8.5 Répartition des âges selon le genre et le statut marital - Chefs de ménage et conjoints uniquement . . . 77
8.6 Répartition des âges des chefs de ménages célibataires . . . 78
8.7 Répartition des âges selon le genre et le statut marital - Individus de 15 ans et plus . . . 79
8.8 Répartition des individus célibataires de 15 ans et plus . . . 79
8.9 Répartition des mariages selon l'âge des conjoints – Modèle 1 . 80
8.10 Répartition des mariages selon l'âge des conjoints – Modèle 2 . 81
8.11 Répartition des mariages selon l'âge des conjoints – Modèle 3 . 81
8.12 Répartition des mariages selon l'âge des conjoints – Modèle 3 . 83

D.1 Diagramme des classes pour VirtualBelgium . . . 117

Introduction

Ce mémoire se place dans le cadre du projet VirtualBelgium sur lequel travaillent L. Hollaert et J. Barthélemy.

Une population synthétique étant construite, nous voulons la faire évoluer dans le temps, notamment en simulant les événements qui peuvent survenir tels que les naissances, les morts, les mariages, les divorces ou encore les déménagements.

Le but de ce mémoire est de modéliser les mariages et les divorces et ensuite d'implémenter les modèles dans le projet VirtualBelgium.

Nous commencerons par introduire le projet VirtualBelgium, élément central de ce mémoire. Nous en donnerons un aperçu général et entrerons ensuite progressivement dans les détails via la description des librairies utilisées et du fonctionnement du programme en lui-même.

VirtualBelgium étant une plateforme de simulation multiagents, nous passerons ensuite à une brève introduction aux systèmes multiagents.

Nous pourrons ensuite passer au coeur de ce mémoire : les différents modèles. Trois types de modèles seront présentés et nous consacrerons un chapitre à chacun d'entre eux.

Le premier modèle est un modèle de choix discrets. Nous réaliserons donc une rapide introduction aux méthodes de choix discrets pour ensuite pouvoir décrire le modèle en lui-même. Le deuxième modèle a pour ambition de déterminer le futur époux d'un individu selon la pression sociale pesant sur ses épaules ainsi que selon son âge. Nous verrons comment définir une approximation de la pression sociale et comment l'âge peut influencer notre

recherche d'un partenaire. Enfin, nous analyserons différentes heuristiques de recherche d'un partenaire se voulant très simples. Nous verrons l'impact qu'a le caractère mutuel de la recherche d'un conjoint.

Le premier modèle ne définissant que si un individu désire se marier et non le partenaire qu'il souhaiterait épouser, nous discuterons du problème des mariages stables permettant de créer des couples satisfaisant certaines propriétés. Pour pouvoir appliquer les algorithmes proposés, nous devrons pouvoir définir le potentiel attractif d'un individu. Nous présenterons donc également un bref aperçu de la littérature portant sur les préférences humaines en matière de conjoint.

Enfin, nous terminerons avec la partie plus technique de ce mémoire. Dans un premier temps, nous résumerons les principes généraux de la suite Repast HPC responsable de la gestion des agents dans VirtualBelgium. C'est alors que nous pourrons expliquer notre implémentation des modèles, les difficultés rencontrées et les solutions pour lesquelles nous avons opté. Nous présenterons alors les résultats numériques obtenus par la simulation de nos différents modèles que nous pourrons comparer à des données réelles.

Chapitre 1

Présentation de VirtualBelgium

La plateforme de simulation VirtualBelgium a pour ambition de comprendre l'évolution d'une population via une simulation de différents aspects : démographie, choix résidentiel, mobilité, ... Ce projet est mené par le Groupe de Recherche sur les Transports (GRT-naXys) et constitue une partie de la thèse de J. Barthélemy[6]. Cet outil est actuellement appliqué à la Belgique.

Dans cette section, nous expliquons les différents éléments qui composent la plateforme mais également les différentes données nécessaires à son utilisation. Á partir de maintenant, nous considérons seulement l'outil appliquée à la Belgique.

1.1 Génération d'une population synthétique

Avant de pouvoir simuler une quelconque évolution avec l'outil VirtualBelgium, ce dernier a besoin d'une population initiale. Malheureusement, pour des raisons de confidentialité et budgétaires, il n'est pas possible de récupérer les données personnelles attendues concernant chaque citoyen belge. Il faut donc générer une population synthétique qui doit être aussi statistiquement proche que possible de la population réelle belge.

Cette population synthétique a été générée par J. Barthélemy et Ph. Toint selon une nouvelle méthode qu'ils proposent dans l'article [5]. Celle-ci est composée de 10 262 160 individus répartis en 4 236 202 ménages dans les 589 communes belges. Les données utilisées dans le processus concernent la

population belge en 2001 et proviennent des sources suivantes :

- *Directorate-general Statistics and Economic information* du gouvernement fédéral belge (2001)
- *Service public fédéral Mobilité et Transports* du gouvernement fédéral belge (2000)
- *GéDAP* [1] centre de l'université catholique de Louvain(2001)
- L'enquête nationale de mobilité *MOBEL* [21]

Ensuite, dans VirtualBelgium, l'évolution de la population synthétique est prise en charge par un programme implémenté en C++ dont la structure est présentée à la figure 1.1. Ce programme simule une évolution temporelle (démographie) et spatiale (mobilité) de la population synthétique, variant selon les données qui lui sont fournies et utilisant le framework Repast HPC, présenté dans la section suivante.

Après l'exécution du programme, des outputs tels que des statistiques concernant l'évolution ou encore des fichiers contenant les déplacements des individus sont créés. Ils sont ensuite exploités par le logiciel MATSim qui permet une visualisation du trafic routier. Ce logiciel sera également présenté dans la section suivante.

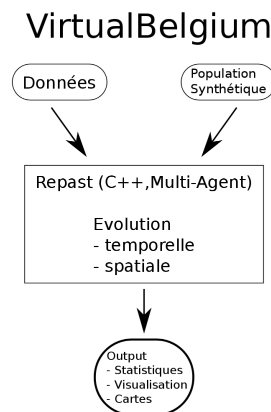

FIGURE 1.1 – Structure de VirtualBelgium

1. Groupe d'étude de démographie appliquée

La suite de ce chapitre permet de comprendre l'architecture du programme VirtualBelgium lié à la plateforme. Il est conseillé de connaitre certaines notions des langages de programmation orientée objet. L'annexe **??** donne, si nécessaire, quelques notionsde base à ce sujet.

1.2 Les logiciels utilisés

Comme nous venons de le dire dans la section précédente, les deux logiciels principalement utilisés par le programme VirtualBelgium sont Repast HPC et MATSim. Leur utilisation et leurs principales caractéristiques sont décrites ci-dessous.

1.2.1 Repast HPC

Repast HPC (REcursive Porous Agent Simulation Toolkit for High Performance Computing) est un framework C++ utilisé pour la simulation et la modélisation de systèmes multi-agents. D'après la référence [1], on appelle multi-agents, un système où plusieurs agents évoluent et qui possède des caractéristiques telles que le parallélisme [2], la robustesse [3] et l'extensibilité [4]. Cependant, le terme « agent » peut être interprété de plusieurs manières en informatique. Dans notre cas, un agent est un système informatique situé dans un environnement qu'il peut modifier, qui agit de manière autonome et flexible pour atteindre les objectifs qui lui sont fixés. Les agents sont capables d'interagir entre eux.

Repast est un outil gratuit, open source et une plate-forme de modélisation et simulation multi-agents. Cet outil est surtout utilisé pour des applications de sciences sociales. À l'origine, il a été développé par David Sallach

2. "En informatique, le parallélisme consiste à implémenter des architectures d'électronique numérique permettant de traiter des informations de manière simultanée, ainsi que les algorithmes spécialisés pour celles-ci. Ces techniques ont pour but de réaliser le plus grand nombre d'opérations pour réduire le temps d'exécution." Cette définition provient du site de référence [42].

3. "La robustesse d'un programme est entre autres, sa capacité à bien fonctionner". Cette définition est extraite du site de référence [11].

4. "Un système est extensible si on peut étendre ses capacités par ajout de mises à jour ou de modules." Cette définition est extraite du site de référence [11].

ainsi que d'autres chercheurs de l'Université de Chicago et du Laboratoire National D'Argonne. Désormais, il est dirigé par l'organisation de bénévoles ROAD (Repast Organization for Architecture and Development).

Son architecture permet de représenter chaque agent par un identifiant unique et prend en charge la gestion des opérations parallèles grâce aux librairies MPI et Boost. Dans VirtualBelgium, les individus et les ménages de la population synthétique sont représentés par des agents.

Dans l'annexe D reprenant un manuel d'utilisation de VirtualBelgium, les indications pour l'installation de Repast HPC sont reprises de même que des informations relatives à la compilation et au débugage dans cet environnement.

1.2.2 MATSim

MATSim (Multi-Agent Transport Simulation) est un outil de simulation de transport. Il s'agit d'un projet open source principalement créé par les groupes suivants :

- *Transport Systems Planning and Transport Telematics*, Institute for Land and Sea Transport Systems, Technische Universität Berlin
- *Transport Planning*, Institute for Transport Planning and Systems, Swiss Federal Institute of Technology Zurich
- *Senozon*, entreprise suisse avec une filiale en Allemagne

Dans VirtualBelgium, MATSim est utilisé pour créer le réseau utilisé par le programme, à partir des données extraites du site OpenStreetMap [3]. *OpenStreetMap est une carte du monde librement modifiable, collaborative où les données sont libres d'être téléchargées et utilisées, sous les termes d'une licence ouverte.* Une section sera consacrée au réseau à la fin de ce chapitre.

1.3 Cœur du programme

Regardons à présent l'architecture du programme VirtualBelgium. Celle-ci est représentée en annexe, à la figure D.1, qui décrit les différentes classes

implémentées. Une classe est un principe utilisé dans la programmation orientée objet. Elle est constituée d'un ensemble d'attribut et de fonctions, appelées méthodes.

Sur cette figure, chaque cadre correspond à une classe du programme dont le nom est inscrit en haut de ce cadre. Ce dernier est divisé en deux parties : la première contient les variables de la classe et la deuxième les méthodes principales. De plus, des liens indiquent les relations éventuelles existantes entre les différentes classes.

Parcourons les différentes classes et résumons leurs rôles et leurs principales caractéristiques.

1.3.1 Classes générales

La classe `Data` est chargée de lire et récupérer toutes les données nécessaires au programme VirtualBelgium. Les données utilisées sont les suivantes :

- *propriétés des modèles* telles que l'évolution spatiale ou temporelle ;
- *données socio-démographiques* : pyramide des âges par commune pour les hommes et les femmes, probabilité de naissance par âge, probabilité de mort par âge et sexe, probabilité d'avoir un garçon ou une fille ;
- *données relatives au réseau* : noeuds représentant chaque commune et réseau routier ;
- *données relatives au modèle d'activités* : codes pour les activités, paramètres de distribution concernant les activités (distance, durée du voyage, durée de l'activité et heure de départ et d'arrivée).

La classe `RandomGenerators` contient des générateurs [5] de nombres pseudo-aléatoires suivant une loi de probabilité, utilisés tout au long du programme.

La classe `Model` se charge principalement de l'initialisation du modèle et décrit la procédure d'évolution que subit la population via la méthode `step`.

5. Un générateur de nombres pseudo-aléatoires est un outil permettant de ressortir une suite de nombre suivant une loi demandée à partir d'un nombre qu'on appelle graine (seed). Le générateur n'est pas valide à 100%. En effet, deux graines identiques dans des générateurs identiques produiront deux suites de nombres identique.

La méthode `step` est utilisée pour itérer sur tous les agents du modèle afin qu'ils accomplissent les tâches demandées par le modèle, telles que vieillir, mourir, déménager, se rendre à leurs activités de la journée, se marier ou divorcer, ...

1.3.2 Individus et Ménages

VirtualBelgium comporte deux différents types d'agents : les ménages représentés par la classe `Household` et les individus par la classe `Individual`.À chaque agent sont associés plusieurs variables ainsi qu'un identifiant unique, déterminé par le logiciel Repast.

Un ménage contient les identifiants de chaque individu le composant, l'identifiant du noeud du réseau correspondant au domicile du ménage ainsi que le code INS [6] de sa commune. La classe `Household` contient également des attributs concernant le nombre d'enfants (de 0 à 5), le nombre d'adultes supplémentaires (de 0 à 2) et le type de ménage parmi :

- Homme seul sans enfant
- Femme seule sans enfant
- Homme seul avec enfant(s)
- Femme seule avec enfant(s)
- Couple sans enfant
- Couple avec enfant(s)

Un individu est décrit par son genre (féminin ou masculin), son âge et la classe d'âge associée, sa catégorie professionnelle (actif, inactif ou étudiant), son niveau d'éducation (aucun diplôme, diplôme primaire, diplôme secondaire ou diplôme supérieur), l'identifiant du ménage dont il fait partie et la place qu'il y occupe (chef de ménage, conjoint, enfant ou adulte supplémentaire) ainsi que son obtention ou non du permis de conduire.

De plus, les individus possèdent également une chaîne d'activités, c'est-à-dire une liste d'activités qu'ils doivent effectuer pendant leur journée. Par exemple, l'agenda d'un étudiant peut se résumer à une activité (aller à

6. Le code INS est un code à 5 chiffres attribué à chaque commune par l'Institut National de la Statistique. Cette définition provient de la référence [32]

l'école) nécessitant deux déplacement : le premier étant d'aller à l'école et le deuxième, de retourner au domicile.

Les différentes activités possibles sont les suivantes :

- Déposer, reprendre quelqu'un
- Activités à la maison
- Travail
- École
- Manger à l'extérieur
- Faire des courses
- Activités personnelles (banque, docteur)
- Rendre visite à la famille ou aux amis
- Promenade
- Loisirs
- Autre
- Aucune

Toute activité d'un individu se voit attribuer un lieu et une durée. Une telle modélisation par chaines d'activités permet de simuler les déplacements des individus dans le réseau, en d'autres mots leur évolution spatiale. Les chaînes d'activités nous permettent donc d'analyser la mobilité belge en vue de mieux la comprendre.

1.3.3 Réseau

Le réseau utilisé dans VirtualBelgium est le réseau routier belge entier constitué de tous les carrefours et routes du pays et extrait du site OpenStreetMap. Le réseau est ensuite modifié par OSMOSIS [7] pour conserver uniquement les données nécessaires au programme VirtualBelgium correspondant au réseau routier belge avec comme conséquence, par exemple, que les noeuds représentant les gares dans OpenStreetMap ne sont pas pris en considération. Ensuite, ce réseau est converti dans un type de fichier XML

7. OSMOSIS est une application java pour la manipulation des données OSM, extraites d'OpenStreetMap.

exploitable par VirtualBelgium et MATSim.

Dans le programme, le réseau est représenté par un *graphe*. Un graphe est un objet mathématique constitué de deux ensembles finis : le premier, noté V, contient n points appelés *noeuds* ou *sommets* labellisés de 1 à n ; le deuxième, noté E, contient des liens entre certaines paires de noeuds. Un lien entre deux noeuds est appelé *arc* et est noté (i, j) lorsqu'il relie les noeuds i et j.

Notre réseau est constitué d'environ 262.000 noeuds et 830.000 arcs. Les noeuds et arc représentent respectivement les carrefours du réseau et les routes. Les arcs ne sont pas orientés.

1.4 Programmation parallèle

Vu la quantité de données traitées dans VirtualBelgium avec ses 10 262 160 individus et ses 4 326 202 ménages, ce programme ne peut s'exécuter sur une unique machine possédant un seul processeur. En effet, si on exécute le programme uniquement en se restreignant à la population de Namur sur un ordinateur personnel avec deux processeurs, il faut déjà attendre deux jours pour obtenir les résultats.

Une solution à ce problème est la programmation parallèle, traitable par le logiciel Repast et par la norme MPI. Ce type de programmation consiste à effectuer différentes tâches en même temps grâce à l'utilisation de plusieurs processeurs [8]. L'utilisation de la norme MPI (Message Passing Interface) permet d'exploiter les ordinateurs multiprocesseurs en assignant l'exécution d'une suite séquentielle d'opérations à chaque processeur. Cette assignation est réalisée en utilisant des opérateurs d'envoi et de réception de messages. Chaque processeur possède son propre espace mémoire. Si on veut une interaction entre les processeurs, il existe des messages spécifiques relatifs à la synchronisation.

8. Un processeur est la partie centrale d'un ordinateur qui effectue les opérations arithmétiques et logiques. Cette définition a été inspirée du site de FUTURA- SCIENCES [15].

Dans le programme VirtualBelgium, chaque processeur va posséder son propre ensemble de ménages et donc son propre ensemble d'individus composant ces ménages. Actuellement, aucune synchronisation n'est effectuée entre ces processeurs, il n'y a donc pas de lien entre les ménages de chaque processeur. Le réseau de VirtualBelgium n'est pas commun. En effet, chaque processeur possède son propre réseau, ce qui veut dire que si on modifie un noeud dans un processeur, il n'est pas modifié dans le réseau des autres processeurs.

En conclusion, il faut garder en mémoire le fait que le programme VirtualBelgium est adapté à la programmation parallèle et que chaque processeur possède son propre ensemble d'individus et son propre réseau.

1.5 Conclusion

En conclusion, le programme VirtualBeglium possède une multitude d'outils, autant pour comprendre l'évolution de la démographie que pour comprendre les déplacements effectués par les habitants de la Belgique.

Cette introduction au projet VirtualBelgium a été écrite en collaboration avec W. Henrotin, S. Marchal et E. Ramelot.

Chapitre 2

Introduction aux systèmes multiagents

Comme nous l'avons mentionné dans le chapitre précédent, VirtualBelgium utilise une modélisation multiagents et nous tentons donc dans ce chapitre d'introduire ce concept.

De nos jours, le concept d'agent est vaste et utilisé dans de nombreux domaines. C'est pourquoi, nous devons tout d'abord définir ce qui est appelé « agent » dans le contexte des systèmes multiagents.
Jennings, Sycara et Wooldridge [23] nous donnent la définition suivante :

> *Un agent est un système informatique, situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu.*

Les mots clés de cette définition sont *situé*, *autonome* et *flexible*. Nous allons donc détailler ces notions.

- On dit d'un agent qu'il est *situé* dans un environnement lorsqu'il reçoit des informations sur son environnement et est capable d'agir sur ce dernier en fonction de ces informations.
- L'agent est *autonome* s'il fait ses propres choix sans intervention extérieure, c'est-à-dire sans l'intervention d'un humain ou d'un autre agent.
- Chaib-draa, Jarras et Moulin [9] nous décrivent le côté *flexible* comme comportant trois caractéristiques :

- ▷ *capable de répondre à temps* : l'agent est capable de percevoir son environnement et élaborer une réponse dans les temps requis ;
- ▷ *proactif* : l'agent exhibe un comportement proactif et opportuniste et est capable de prendre l'initiative au bon moment ;
- ▷ *social* : l'agent est capable d'interagir avec les autres agents.

Ces différentes propriétés doivent cependant être nuancées selon les applications, qui peuvent en accentuer certaines ou en oublier d'autres.
Un système multiagent (SMA) est un système composé d'un ensemble d'agents qui interagissent ou simplement coexistent.

Les interactions peuvent être de plusieurs types dont les plus courants sont la coopération, la coordination et la négociation. Les agents peuvent aussi interagir de plusieurs façons : par communication directe, par communication via un agent intermédiaire ou en modifiant leur environnement.

Dans le cadre de VirtualBelgium, les agents d'un même ménage doivent se coordonner pour les différentes activités de la journée. Un individu doit également planifier sa journée et, par exemple, choisir son heure de départ pour le travail en fonction du trafic. Dans ce cas, il n'y a pas de communication directe mais il y a bien interaction au travers de la présence des autres agents sur les routes, une modification de l'environnement.

Jennings, Sycara et Wooldridge [23] nous indiquent que les caractéristiques principales des SMA sont les suivantes :

- chaque agent a des informations ou capacités limitées ainsi qu'un point de vue partiel ;
- il n'y a pas de contrôle global du système multiagent ;
- les données sont décentralisées ;
- le calcul est asynchrone.

Chaib-draa, Jarras et Moulin [9] pensent que la technologie agent pourrait devenir un nouveau paradigme de programmation similaire à la *programmation orientée objet* qu'ils suggèrent d'intituler *programmation orientée agent*. Cependant, même s'ils indiquent la ressemblance entre *objet* et *agent*, ils insistent également sur leurs différences. Tout comme les objets, les agents encapsulent leur état interne qui peut être communiqué ou modifié via leurs méthodes. Ils se différencient cependant par leur autonomie. En effet, une

méthode d'un objet doit être appelée par un autre objet pour s'effectuer alors qu'un agent recevra une requête et *choisira* d'agir ou non.
Une autre différence est le caractère *flexible* des agents qui n'est pas prévu par le paradigme orienté objet.
Enfin, une dernière distinction à pointer est la capacité des agents à être source de contrôle dans leur environnement, capacité que n'ont pas les objets puisqu'il n'y a jamais qu'une seule source de contrôle dans un système orienté objet.

Si l'utilisation de systèmes multiagents est très intéressante, leur implémentation peut se révéler être un vrai défi. Heureusement, des outils existent pour nous rendre l'implémentation plus aisée et l'un d'eux, Repast, est le sujet de notre prochain chapitre.

Chapitre 3

Première modélisation des mariages et divorces

Notre premier modèle est celui implémenté par J. Barthélemy ,Ch. Hubaux, A. Fuzfa, R. Lambiotte et Ph. Toint dans le cadre du projet multidisciplinaire donné en 2010 - 2011.

Le principe de cette modélisation est de diviser le choix en deux choix discrets successifs. Tout d'abord, l' individu décide s'il souhaite être célibataire ou en couple. Plus précisément, si l'individu est actuellement marié, il pourra rester marié ou divorcer ; s'il est célibataire, il pourra alors choisir entre le rester ou se marier. Ensuite, s'il choisit de se marier, il peut préférer se mettre en couple avec une personne ayant des enfants ou une personne sans enfants.

Nous commencerons donc ce chapitre par une introduction aux choix discrets basée sur les sources [4] et [39]. Ensuite, nous détaillerons chacun des deux choix et terminerons par quelques remarques sur ce premier modèle.

3.1 Introduction aux méthodes de choix discrets

Le but de la théorie des choix discrets est de modéliser le choix d'un individu parmi un ensemble fini d'alternatives à partir de caractéristiques de cet individu.

L'ensemble des choix doit vérifier certaines contraintes : il doit être exhaustif, fini et les différents choix qui le composent doivent être mutuellement exclusifs.

La modélisation du choix se fait à travers des fonctions d'utilité U_j définies pour chaque alternative j. Une alternative i est alors choisie par un individu n si

$$U_{ni} > U_{nj} \quad \forall j \neq i$$

Évidemment toutes les caractéristiques influençant le choix d'un individu ne peuvent pas être connues, la fonction d'utilité est donc séparée en une fonction V_j dépendant uniquement des caractéristiques connues d'une part et une partie non observée ϵ, aussi appelée *erreur*, d'autre part.

On a alors

$$U_{nj} = V_{nj}(x_{nj} + s_n) + \epsilon_{nj}$$

où x_{nj} dénote les caractéristiques observées de l'alternative j pour l'individu n et s_n correspond aux caractéristiques propres à l'individu n.

De plus, comme certaines caractéristiques sont inconnues, le choix de l'individu ne sera pas déterminé exactement mais on calculera la *probabilité* qu'il fasse ce choix.

Ainsi, la probabilité P_{ni} que l'individu n choisisse l'alternative i est définie comme

$$\begin{aligned}
P_{ni} &= P(U_{ni} > U_{nj} \quad \forall j \neq i) \\
&= P(V_{ni} + \epsilon_{ni} > V_{nj} + \epsilon_{nj} \quad \forall j \neq i) \\
&= P(\epsilon_{nj} - \epsilon_{ni} < V_{ni} - V_{nj} \quad \forall j \neq i) \\
&= \int I[\epsilon_{nj} - \epsilon_{ni} < V_{ni} - V_{nj} \quad \forall j \neq i] \; f(\epsilon_n) \; d\epsilon_n
\end{aligned}$$

où $f(\epsilon_n)$ est la fonction de densité de ϵ_n et $I(.)$ est la fonction indicatrice, c'est-à-dire qu'elle vaut 1 lorsque la condition entre crochets est vérifiée et 0 dans le cas contraire.

Une remarque importante à noter est que seul le signe de la différence entre les utilités importe, sa valeur absolue n'est pas utilisée. De même, l'échelle de l'utilité n'est pas non plus importante, ajouter une même constante aux utilités de toutes les alternatives ne changerait en aucun point les résultats.

3.2 Choix entre mariage et célibat

Pour ce premier choix, nous considérons deux fonctions d'utilité : V_s l'utilité d'être célibataire et V_c l'utilité d'être en couple. Ces fonctions d'utilité tiennent compte des attributs suivants :

- l'âge,
- le sexe,
- l'historique du statut matrimonial (nombre de changements du statut, temps écoulé depuis le dernier changement et statut actuel),
- le statut socio-professionnel.

Pour plus de clarté, nous détaillons ici l'effet de chaque paramètre séparément, nous noterons les changements d'utilité apporté par un paramètre Δ_s et Δ_c pour les utilités respectives V_s et V_c.

Âge

Commençons par définir f_{age}, une fonction de l'âge qui sera utilisée pour modifier les utilités. Notons x, l'âge de l'individu moins 18 ($x = age - 18$).

$$f_{age} = 6\frac{2}{\sqrt{3\sigma\sqrt{\pi}}}\frac{1-x^2}{\sigma^2}\mathrm{e}^{\frac{-x^2}{2\sigma^2}} + 1.2;$$

où $\sigma = 10$.
Nous pouvons désormais définir les changements d'utilité :

$$\Delta_s = -f_{age} + 2 \quad \text{et} \quad \Delta_c = f_{age}$$

Sous cette forme, ces changements peuvent être difficiles à interpréter, traçons donc les valeurs de Δ_s (en bleu) et Δ_c (en vert) en fonction de l'âge.

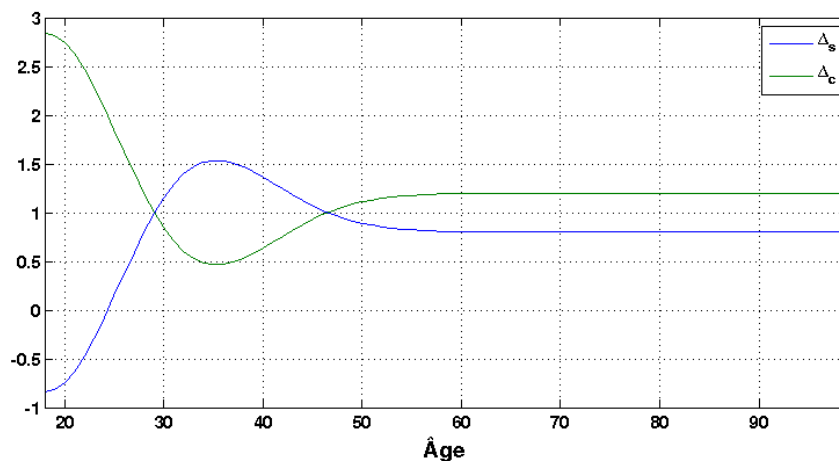

FIGURE 3.1 – Graphe de f_{age} en fonction de l'âge

Ces résultats sont surprenants et ne correspondent pas tout à fait à ce que pourrions imaginer, et cela surtout pour les personnes très jeunes qui préfèreraient se marier qu'être célibataires. Néanmoins, l'âge n'est pas la seule variable prise en compte et est complété par d'autres paramètres.

Sexe et âge

Intervient alors un croisement entre les variables sexe et âge. Si l'individu est un homme de moins de 25 ans, son utilité V_s est augmentée de 1. À l'inverse, s'il s'agit d'une femme de moins de 30 ans, c'est l'utilité V_c qui est augmentée de 2.
Ce choix est assez intuitif, les jeunes hommes souhaitent généralement rester célibataires tandis que les jeunes femmes sont peut-être plus pressée de trouver un conjoint.

Statut socio-professionnel et âge

Cette section modifie uniquement les utilités des étudiants.
Si l'étudiant a moins de 25 ans, $\Delta_s = -\Delta_c = 2.5$.
S'il est plus âgé, alors les modifications sont moins importantes mais présentes : $\Delta_s = -\Delta_c = 1$.
Encore une fois, ce choix se comprend facilement puisqu'il est plus rare de se

marier lorsque l'on est encore étudiant.

Ce paramètre compense le problème survenu dans la section **Âge**. En effet, les individus de moins de 25 ans sont souvent étudiants et, dans ce cas, en ne tenant compte que des paramètre **Âge** et **Statut socio-professionnel**, ils choisiront plutôt de rester célibataires.

Situation matrimoniale et âge lors de son dernier changement

On définit une fonction f_{stay}. Pour cela, notons age_{change} l'âge de l'individu lors de son dernier changement de situation.

$$f_{stay} = \frac{1 + \tanh(age_{change} - 4.5)}{2}$$

Comme pour f_{age}, traçons f_{stay} en fonction de age_{change}.

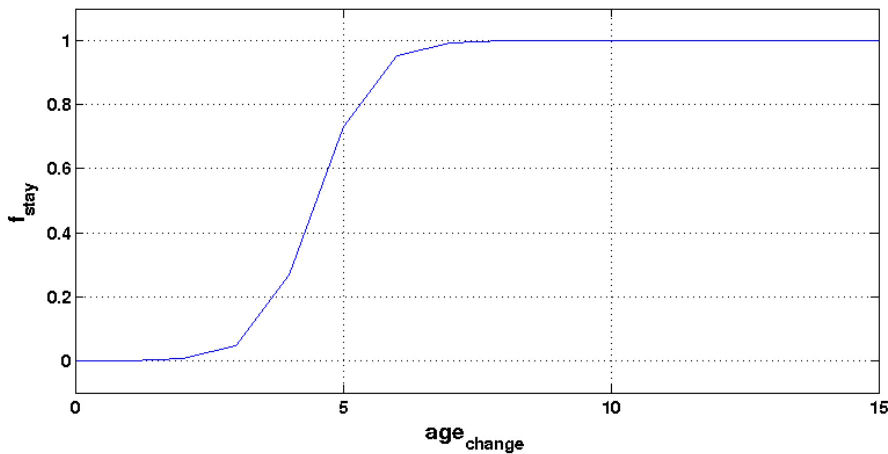

FIGURE 3.2 – Graphe de f_{stay} en fonction de age_{change}

Nous avons alors les changements d'utilités suivants : si l'individu est célibataire $\Delta_s = f_{stay}$ et $\Delta_c = -f_{stay}$; si l'individu est en couple, la situation est inversée et on obtient $\Delta_s = -f_{stay}$ et $\Delta_c = f_{stay}$.

Cette intervention peut se comprendre comme une tendance à conserver la situation matrimoniale actuelle. Au plus de temps un individu est resté dans une même situation, au plus il s'y est habitué et donc au moins il désire en changer.

Nombre de changements du statut matrimonial

On peut supposer que le nombre de changements de situations, c'est-à-dire le nombre de divorces vécus, dissuade une personne de se marier. On définit alors les changements d'utilité par

$$\Delta_s = n_{change} \quad \text{et} \quad \Delta_c = -n_{change}$$

où on note n_{change} le nombre de changements.

Tous ces différents paramètres doivent ensuite être combinés et leurs effets sont sommés pour obtenir les utilités finales décidant du choix de l'individu.

3.3 Choix du partenaire : seul ou avec enfants

Ce choix tient compte de deux caractéristiques de l'individu : son âge et le nombre d'enfants qu'il a.
Notons V_e l'utilité de choisir un partenaire ayant des enfants et V_n celle de choisir un partenaire seul. De nouveau, utilisons les notations Δ_e et Δ_n pour les changements d'utilité apporté par chaque paramètre.

Âge

Les utilités tiennent compte du logarithme naturel de l'âge :

$$\Delta_e = -\ln(\text{âge}) + 7 \quad \text{et} \quad \Delta_n = -\ln(\text{âge})$$

Ces fonctions ont les graphes suivants :

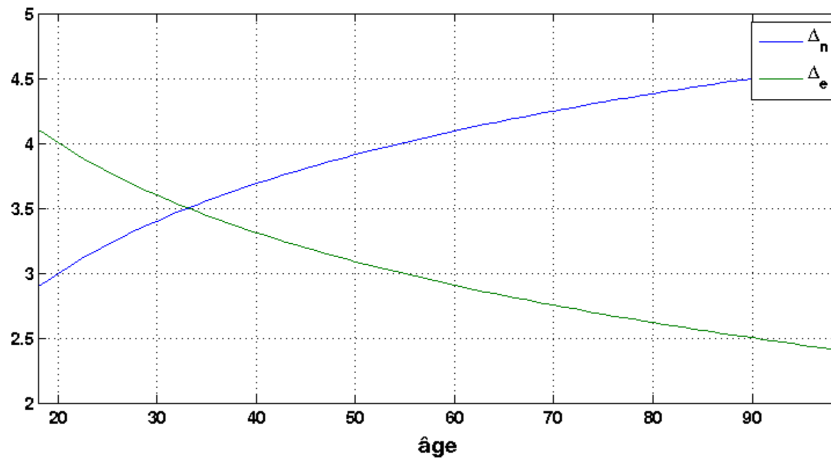

FIGURE 3.3 – Graphes de Δ_e et Δ_n en fonction de l'âge

À première vue, ces graphes ne paraissent pas convaincants puisqu'ils semblerait que les jeunes chercheraient plus à se mettre en couple avec des personnes ayant déjà des enfants, ce qui n'est pas réaliste. Cependant, il faut ici prendre les deux paramètres en compte pour se faire une idée correcte de leur effet. Définissons donc l'effet du nombre d'enfants sur les utilités et nous regarderons aux graphes de leur effet conjoint ensuite.

Nombre d'enfants

Notons n le nombre d'enfants, alors

$$\Delta_e = \frac{n}{2} - 1 \quad \text{et} \quad \Delta_n = -\frac{n}{2} + 1$$

Dans VirtualBelgium, le nombre d'enfants est limité à 5, regardons donc les graphes des utilités V_e (en vert) et V_n (en bleu) en fonction de l'âge pour chacun des cas.

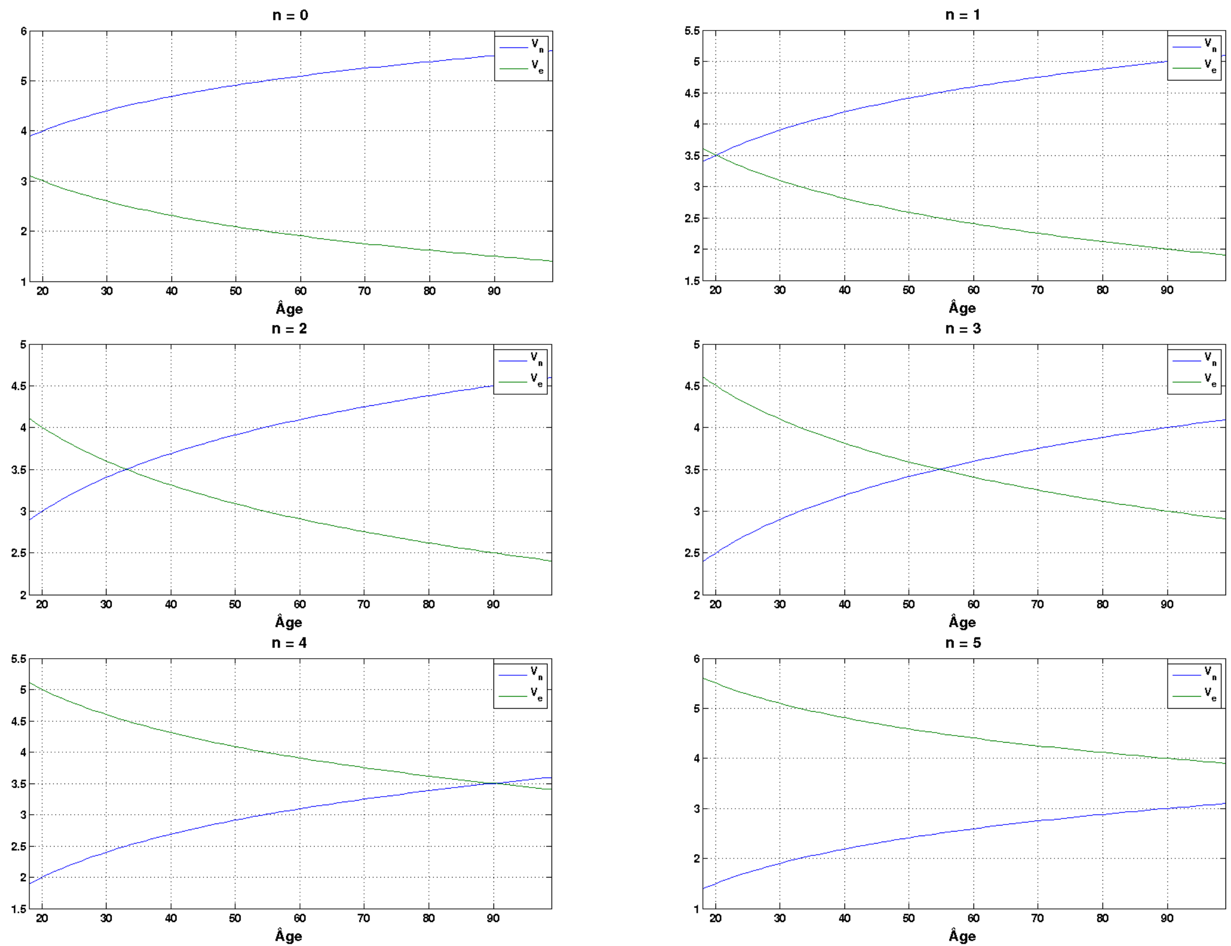

FIGURE 3.4 – Graphes des utilités V_e et V_n en fonction de l'âge selon le nombre d'enfants n

Nous observons que, dans ce modèle, les personnes n'ayant pas d'enfants ne désirent pas se mettre en couple avec une personne ayant déjà des enfants ; les personnes ayant 2 enfants préfèrent trouver un conjoint dans la même situation qu'eux lorsqu'elles sont suffisamment jeunes et à partir de 3 enfants, on préfère également rencontrer une personne ayant des enfants.

Chapitre 4

Problème des mariages stables

4.1 Introduction

Une fois une liste de célibataires à la recherche d'un conjoint obtenue, il faut ensuite pouvoir tenir compte des préférences de chacun afin de former des couples réalistes. Pour cela, nous nous sommes intéressés à un problème très connu et étudié dans la littérature : le problème des mariages stables. Ce problème s'adapte assez bien à notre problématique mais nécessitera quelques modifications.

Dans ce chapitre, nous analyserons dans un premier temps la théorie du problème. Ensuite, nous détaillerons les algorithmes proposés pour sa résolution du problème classique et de quelques extensions, et nous parlerons brièvement de leur complexité.

Enfin, nous aborderons le sujet des préférences des individus en ce qui concerne leurs partenaires afin de pouvoir définir une méthode de « quotation » qui servira au calcul des liste de préférences de chacun.

4.2 Problème classique

Le problème des mariages stables consiste en la formation de couples entre n hommes et n femmes. Le couplage obtenu doit alors être considéré satisfaisant aux termes du problèmes, c'est-à-dire *stable*. Nous pouvons poser les bases du problème en nous basant sur [27].

Définition 4.2.1. *Soient H et F deux ensembles finis de n éléments où $H = \{A_1, A_2, ..., A_n\}$ est l'ensemble des hommes à marier et $F = \{a_1, a_2, ..., a_n\}$ l'ensemble des femmes à marier. Un couplage est une bijection de H sur F représentant un ensemble de n couples formés entre les hommes et les femmes.*

Supposons l'existence d'un ordre de préférence pour chaque homme et chaque femme. Cela revient à dire que, pour chaque homme, on peut établir la liste des n femmes partant du premier choix au dernier, et qu'une telle liste existe similairement pour chaque femme. Nous pouvons donc reporter ces préférences dans deux tableaux $n \times n$. Nous supposerons de plus que l'ordre de préférence est strict, cela signifie qu'un individu n'est jamais indifférent lors d'un choix entre deux partenaires potentiels.

Définition 4.2.2. *Un couplage est instable si un homme A et une femme a, non mariés entre eux, se préfèrent mutuellement à leurs partenaires. Nous appellerons une telle paire (A, a) une paire instable.*

Un couplage est donc considéré comme instable lorsque la situation crée une opportunité de liaison. Cette notion de stabilité est donc assez intuitive.

Exemple 4.2.1. *Illustrons notre problème à l'aide d'un cas très simple comprenant trois hommes A, B, C et trois femmes a, b, c. Nous pouvons lister les préférences de chacun dans les tables suivantes.*

Homme	Ordre de préférence
A	b c a
B	c a b
C	a b c

Femme	Ordre de préférence
a	A B C
b	B C A
c	C A B

Dans cet exemple, il existe six couplages différents dont trois sont stables et les trois autres instables. Le premier couplage stable que nous pouvons trouver est de marier chaque homme à son premier choix. Une deuxième solution serait, à l'inverse, de mettre en couple chaque femme et son premier choix. Une dernière possibilité pour obtenir un couplage stable est que chacun

épouse son deuxième choix.

La stabilité des deux premières propositions de couplages est triviale puisque soit aucune homme, soit aucune femme, n'aurait préféré quelqu'un d'autre à son partenaire actuel.

Vérifions la stabilité de la dernière alternative qui crée donc les couples suivants : $\{(A, c), (B, a), (C, b)\}$*. Pour cela, nous pouvons établir pour chaque homme, et chaque femme, la liste des partenaires qu'il, ou elle, aurait préféré épouser.*

Hommes :	*Femmes :*
— *A préférait b*	— *a préférait A*
— *B préférait c*	— *b préférait B*
— *C préférait a*	— *c préférait C*

Ces listes ne se recoupent pas et la stabilité est donc vérifiée.

Nous pouvons prendre un exemple de couplage instable. Supposons que les couples formés soient (A, c),(B, b) *et* (C, a)*. Alors B préfère c et a à sa partenaire actuelle. De même, a préfère A et B à son partenaire actuel. Il y a donc une instabilité puisque B et a se préfèrent mutuellement à leur partenaires actuels.*

La question de l'existence se pose alors. Existe-t-il un couplage stable pour toute disposition de préférences ? Gale et Shapley[16] répondent positivement à cette question et soumettent un algorithme permettant de toujours trouver un ensemble stable de mariages à un problème donné. Nous commencerons ici par présenter la procédure à suivre qu'ils proposent, nous pourrons ensuite en prouver la correction.

Lors de la première étape, chaque homme demande en mariage la femme qu'il préfère. Chaque fille ayant reçu plus d'une demande rejette toutes les demandes sauf celle de l'homme qu'elle préfère parmi ses prétendants. Elle garde donc ce dernier « en réserve » mais n'accepte pas directement son offre au cas où une meilleur offre se présentait.
Ensuite, les hommes ayant été rejetés font leur demande à leur deuxième choix. De nouveau, chaque femme concernée choisit l'homme qu'elle préfère parmi les nouveaux prétendant et celui qu'elle avait gardé en réserve, s'il existe, et rejettent tous les autres. Aucune offre n'est acceptée encore.

Il faut continuer ce processus en descendant dans l'ordre de préférence des hommes et cela jusqu'à ce que chaque femme ait reçu une demande.

Kleinberg et Tardos [26] réécrivent cet algorithme sous la forme suivante :

```
while un homme m est libre et n'a pas demandé toutes les femmes do
    Choisir un tel homme m
    w ⟵ femme la mieux classée dans la liste de m et à qui il n'a pas
    encore fait sa demande
    if w est libre then
        fiancer m et w
    end
    else
        m' ⟵ fiancé actuel de w
        if w préfère m' à m then
            m reste libre
        end
        else
            fiancer m et w
            m' devient libre
        end
    end
end
```

Nous pouvons désormais nous assurer que l'algorithme renvoie bien un couplage stable dans tous les cas.

Tout d'abord, remarquons qu'après la réception de sa première demande, une femme ne redevient jamais libre. En effet, elle peut changer de fiancé si une meilleur offre apparaît mais ne peut pas perdre son fiancé actuel.

En revanche, un homme peut être fiancé à un certain point et redevenir libre au cours de l'algorithme. Cependant, si un homme est libre à un moment de l'exécution, alors il reste au moins une femme à qui il n'a pas fait sa demande. En effet, si un homme m est libre alors une femme w doit l'être aussi puisqu'il y a autant d'hommes que de femmes. Par notre précédente remarque, dire que w est libre signifie qu'elle n'a jamais été demandée en

mariage. Notre homme m n'a donc pas pu demander toutes les femmes en mariage.

Enfin, nous pouvons montrer que le couplage obtenu par l'algorithme est stable.

Démonstration. Supposons, par l'absurde, qu'il existe une instabilité impliquant les couples (m, w) et (m', w') et telle que m préfère w' à w et w' préfère m à m'.

Si w' préfère m à m', elle n'a pas pu rejeter la proposition de m en faveur de celle de m', cela implique donc que m n'a jamais fait sa demander à w'.

Or, la sequence des femmes à qui un homme fait sa demande se fait dans l'ordre de sa liste de préférence et ce, sans sauter d'étape. Donc, si m préfère w' à w, il a dû lui faire sa demande à un moment de l'exécution de l'algorithme.

Nos deux conclusions se contredisent, une telle instabilité n'a donc pas pu ressortir de l'algorithme. □

Concernant la complexité de l'algorithme, nous pouvons dire qu'au plus $n^2 - 2n + 2$ itérations de la boucle `while` seront nécessaires[27].

Concentrons-nous désormais sur la solution construite en elle-même. Sera-t-elle la même pour toute exécution ? En effet, la question se pose suite à la formulation de la première étape de notre algorithme : « *Choisir un tel homme m* ». Le résultat sera-t-il toujours le même selon notre méthode de choix de cet homme m ? La réponse est oui et pour le prouver, Kleinberg et Tardos[26] commencent par caractériser la solution obtenue par l'algorithme Gale-Shapley.

Tout d'abord, remarquons que la solution construite est en fait la solution stable la plus favorable pour les hommes. Une version similaire de l'algorithme où, cette fois, les femmes font leur demande déterminerait, à l'inverse, la solution stable la plus favorable aux femmes. Reprenons notre exemple 4.2.1. Nous avions trois solutions stables, la première mariait chaque homme à sa partenaire préférée ; la deuxième mariait chaque femme à son partenaire favori ; dans la troisième, chacun finissait avec son deuxième choix. Pour un tel problème, l'algorithme G-S générera la première solution s'il est en version hommes, ou la deuxième solution dans sa version femmes. Par

contre, la troisième solution, plus équitable, ne sera pas atteinte avec l'algorithme.

Cela étant dit, formalisons désormais ce que nous appelons *version la plus favorable*. Nous nous concentrons donc désormais sur l'algorithme dans sa version originale, celle où les hommes sont ceux qui font les demandes. Un raisonnement similaire peut évidemment être appliqué pour les versions où les femmes proposent.

Définition 4.2.3. *On appellera une femme w une partenaire valide d'un homme m s'il existe un couplage stable dans lequel m et w sont mariés.*

Définition 4.2.4. *On dira que w est la meilleure partenaire valide de m si w est une partenaire valide de m et qu'aucune femme mieux classée dans les préférences de m n'en est une partenaire valide. Nous noterons $best(m)$ la meilleure partenaire valide de m.*

Nous pouvons désormais décrire le couplage obtenu par G-S comme

$$S^\star = \{(m, best(m)) : m \in M\}$$

Cela siginiferait donc que deux hommes ne peuvent avoir la même meilleure partenaire valide.

Démonstration. Supposons, par contradiction, qu'une exécution ϵ de l'algorithme G-S résulte dans le couplage S différent de $S^\star$. Cela signifie qu'un homme est marié avec une femme qui n'est pas sa meilleure partenaire valide.

Puisque les hommes font leur demande dans l'ordre décroissant de préférence, alors cet homme a été rejeté par sa meilleur partenaire valide durant ϵ. Considérons la première fois qu'un homme valide m est rejeté par une partenaire valide w durant ϵ. Puisque les hommes font leur demande par ordre décroissant de préférence et puisque c'est la première fois qu'un refus de la sorte est rencontré, w devait être la meilleure partenaire valide de m,$best(m)$.

Ce refus signifie que w est fiancée à un homme m' qu'elle préfère à m.

Comme w est une partenaire valide de m, il existe, par définition, un couplage stable S' contenant la paire (m, w). Nous pouvons donc nous demander qui est la partenaire de m' dans ce couplage. Supposons qu'elle soit $w' \neq w$, w' est donc une partenaire valide de m'.

Puisque le rejet de m par w était le premier rejet d'un homme par une de ses partenaires valides lors de l'exécution ϵ, cela signifie que m' n'avait pas été rejeté par une partenaire valide au moment où il s'est fiancé à w. En effet, soit m' s'est fiancé à w avant que m fasse sa demande et soit rejeté, soit m a été rejeté lorsque m' a fait sa demande. Dans les deux cas, m' n'a pas pu être rejeté par une partenaire valide avant de faire sa demande à w sinon le rejet de m ne serait pas le premier de la sorte lors de l'exécution ϵ.

Puisque les demandes se font dans l'ordre de préférence et puisque w' est une partenaire valide de m', m' doit préférer w à w'. Cependant, nous savons déjà que w préfère m' à m. Or, $(m', w) \notin S'$, S' ne peut donc pas être un couplage stable puisque m' et w se préfèrent tous deux à leurs partenaires. Nous obtenons ainsi notre contradiction qui réfute notre hypothèse de départ. □

4.3 Quelques extensions au problème

Dans cette section, nous mentionnons quelques extensions souvent mentionnées dans la littérature concernant le problème des mariages stables. Certains se révéleront plus utiles que d'autres à la résolution de notre problème.

4.3.1 Ensembles de tailles différentes

Si le problème autorise une différence de taille entre les deux ensembles, c'est-à-dire s'il on peut rencontrer une situation où il y a strictement plus d'hommes que de femmes ou inversément, il faut modifier notre définition de stabilité.

Définition 4.3.1. *Une paire $(m, w) \notin S$ est dite bloquante pour S si*

1. *m est sans partenaire ou préfère w à sa partenaire actuelle, et*
2. *w est sans partenaire ou préfère m à son partenaire actuel*

Un couplage S est stable s'il n'admet aucune paire bloquante.

Pour un problème où le nombre d'hommes diffère du nombre de femmes, il existe toujours une solution stable où tous les individus du plus petit groupe sont en couple. Bien entendu, toute solution stable doit marier chaque individu du plus petit groupe.

L'algorithme [1] que nous avons présenté ne nécessite aucune modification puisque la condition de boucle est suffisamment précise. En effet, rappelons que la boucle peut être décrite comme « *tant qu'il existe un homme m libre et n'ayant pas demandé toutes les femmes en mariages* ». Si notre problème contient plus d'hommes que de femmes, alors cela permettra bien à tous les hommes de demander toutes les femmes en mariage. Si les hommes sont moins nombreux, l'algorithme se terminera bien dès que tous les hommes ont été fiancés.

Une propriété intéressante du problème étendu permet de retrouver le problème classique. En effet, il a été montré que le plus grand ensemble d'individu peut être divisé en deux avec d'un côté les individus se retrouvant dans tous les couplages stables et de l'autres ceux qui ne se retrouvent dans aucun[30]. Le problème étendu a donc le même ensemble de couplages stables que le problème classique obtenu en retirant les individus n'étant jamais mariés.

4.3.2 Listes incomplètes

Une autre extension permettant d'ajouter un peu plus de réalisme à notre problème est de permettre aux individus d'en déclarer certains comme inacceptables. On autorise ainsi les listes de préférence incomplètes, les personnes jugées inacceptables sont omises de la liste.

Définition 4.3.2. *On dire que les listes de préférence définies dans un problème sont cohérentes si un homme m apparaît sur la liste d'une femme w si et seulement si w apparaît sur la liste de m.*

La cohérence d'un problème pouvant être aisément forcé en un temps linéraire, nous supposons ici que les listes de préférence sont cohérentes.

Avec cette extension, nous nous retrouvons face à la possibilité de couplages incomplets. La définition de stabilité d'un couplage est alors similaire à celle donnée pour l'extension précédente (Définition (4.3.1)). Une propriété intéressante du problème étendu est qu'à l'instar de notre extension précédente, les ensembles des hommes et des femmes peuvent être divisés en deux avec d'une part les individus mariés dans tous les couplages stables et ceux qui restent célibataires dans tous les mariages stables [17].

4.3.3 Indifférences

Dans le problème classique, nous avons supposé que que l'ordre de préférence est strict. Si nous incluons la possibilité d'indifférence entre deux partenaires potentiels, nous obtenons de nouveaux problèmes. De nouveaux types de stabilité sont introduits [36]. Si ajouter cette possibilité se révèle très intéressant pour de nombreux problèmes, nous ne trouvons pas de réelle nécessité d'une telle extension dans notre situation. Nous ne nous attarderons donc pas sur cette problématique. Cependant, le lecteur intéressé trouvera sans aucun doute l'information nécessaire dans la thèse de S. Scott [36] : « A Study Of Stable Marriage Problems With Ties ».

4.3.4 Un peu plus d'équité

Comme nous l'avons déjà mentionné, l'algorithme Gale-Shapley trouve le mariage stable le plus favorable aux hommes, ou favorisent les femmes si l'on échange les rôles. On dira que le couplage obtenu est *man-optimal* (*woman-optimal* respectivement). Mais ce n'est pas tout, McVitie et Wilson [31] ont observé que le couplage man-optimal marie les femmes à leur pire partenaire valide, nous diront que le couplage est *woman-pessimal*.

Définition 4.3.3. *On dira que m est le pire partenaire valide de w si m est un partenaire valide de w et qu'aucun homme moins bien classé dans les préférences de w n'en est un partenaire valide.*

Théorème 4.3.1. *Le couplage man-optimal marie chaque femme à son pire partenaire valide.*
Autrement dit, il n'existe pas de couplage stable dans lequel une femme se marie avec un partenaire moins bien classé que celui qu'elle épouse dans le couplage man-optimal.

Démonstration. Soit un couplage stable S' et le couplage man-optimal S. Supposons, par l'absurde qu'une femme w préfère son partenaire dans S que dans S'. Appelons ces partenaires respectivement m et m'. Nous savons que m préfère w à toute femme qu'il peut avoir dans un couplage stable par définition de la solution man-optimal. Donc w et m se préfèrent mutuellement à leurs partenaires respectifs dans S', ce qui signifie que S' est instable. □

Cette injustice nous pousse à nous demander s'il n'est pas possible de trouver un algorithme plus égalitaire. En effet, si nous cherchons à être quelque

peu réalistes, nous savons bien que la recherche d'un partenaire dans notre société est mutuelle. Les deux partis doivent être suffisamment heureux de leur partenaire.

Une première méthode serait bien entendu de lister toutes les solutions stables et de choisir la meilleure selon un critère défini. Par exemple, si nous définissons $mp(m, w)$ comme la place de la femme w dans la liste de m, nous pouvons dire que la solution $S^\star$ obtenue par l'algorithme de Gale-Shapley est celle qui minimise

$$\sum_{(m,w)\in S} mp(m, w).$$

Un critère équitable pourrait être la somme de toutes les positions, hommes et femmes. En notant $wp(m, w)$ la place de m dans la liste établie par w, nous pouvons définir ce critère comme

$$\sum_{(m,w)\in S} \left(mp(m, w) + wp(m, w)\right).$$

Il est évident que, de la sorte, les femmes et les hommes sont traités de manière égale. Une telle solution peut être construite via un algorithme de complexité $O(n^{2.5} \log n)$[36].

Une autre optique pour rendre le couplage plus juste est de limiter le *regret* du couplage. Le regret d'une personne dans un couplage est défini comme la position de son partenaire dans sa liste de préférence, on définit alors le regret d'un couplage, noté $r(S)$, comme le regret maximal rencontré dans ce couplage. Le regret d'un couplage s'intéresse donc à la personne la plus malheureuse. Il n'est donc pas question du genre des individus, l'équité est donc atteinte.

Une première solution est attribuée à Selkow, il s'agit d'un algorithme de complexité $O(n^4)$ présenté par Knuth[27]. Cependant, un algorithme moins gourmand, de complexité $O(n^2)$, est par la suite proposé par Gusfield[19]. C'est donc à cette solution que nous allons nous intéresser.

Gusfield décompose le problème en deux sous-problèmes. Le premier sous-problème est de trouver, s'il en existe, un couplage stable de regret minimal

parmi tous les couplages stables dont la personne la plus malheureuse est une femme. Le second sous-problème est le symétrique du premier puisqu'il s'agit de trouver, s'il en existe, un couplage stable de regret minimal parmi ceux dont la personne la plus malheureuse est un homme.

Avant de pouvoir décrire l'algorithme proposé par Gusfield, définissons-en une opération importante, originellement développée dans [31]. Cette opération consiste à séparer le couple d'un homme m dans un couplage stable et de le forcer à épouser une femme moins bien classée dans sa liste.

Définition 4.3.4. *L'opération breakmarriage(S,m) est définie comme : Recommencer l'algorithme Gale-Shapley en séparant m de w, sa partenaire dans S, le rendant ainsi libre, il doit à présent faire sa demande à la femme suivant w sur sa liste. Cette désunion débute donc une nouvelle séquence de fiançailles et de refus comme dans l'algorithme de Gale-Shapley. L'opération se termine lorsque w reçoit une nouvelle demande (rendant ainsi tous les hommes fiancés) ou lorsqu'un homme a été refusé par toutes les femmes.*

Dans la définition de l'opération breakmarriage, deux cas de figure sont présentés. Le premier cas se produit lorsque tous les hommes retrouvent une partenaire, l'opération est alors un succès et le nouveau couplage obtenu est stable[31]. Le second cas correspond à un échec de l'opération, aucun couplage stable n'a pu être trouvé en séparant m et w.

Proposition 4.3.1. *Si m est marié dans S à une femme différente de sa partenaire dans la solution optimale pour les femmes, alors l'opération breakmarriage(m,S) terminera avec succès et retournera donc un nouveau couplage stable.*

L'algorithme, décrit par Gusfield, permet de trouver une solution au problème du regret féminin minimal, en supposant que les deux sous-problèmes aient une solution. Cet algorithme est le suivant :

1. Trouver S_0 le couplage stable man-optimal et trouver le couplage stable woman-optimal $S^\star$. Initialiser i à 0.
2. Soit w une femme de regret $r(S_i)$ dans S_i et soit m son partenaire dans S_i. Si m et w sont en couple dans $S^\star$, alors l'algorithme s'arrête et retourne S_i. Sinon, effectuer l'opération `breakMarriage`(S_i, m) et définir $S_(i+1)$ comme le résultat de cette opération.

3. Si aucune femme dans M_{i+1} n'est de regret $r(M_{i+1})$ alors l'algorithme s'arrête et renvoie M_i. Sinon, il faut incrémenter i et retourner à l'étape 2.

Remarquons tout d'abord que chaque opération breakmarriage est un succès par la proposition (4.3.1). L'algorithme se terminera donc bien puisqu'à chaque opération breakmarriage, une femme changeant de partenaire obtient toujours un meilleur conjoint que le précédent alors qu'un homme changeant d'épouse préférait sa précédent partenaire. Alors à moins que la condition de l'étape 2 soit vérifiée, plus aucune femme n'aura le regret maximal du couplage.

4.4 Caractéristiques recherchées chez un partenaire

Il n'est pas simple de comprendre ou prédire l'attirance d'une personne pour une autre. Différentes études empiriques ont été réalisées sur les préférences amoureuses afin d'examiner les traits que désirent trouver les individus à la recherche d'un conjoint. Des recherches ont été réalisées à propos de la désirabilité perçue de différents traits de caractères ainsi que d'autres attributs tels que l'apparence physique ou le statut social. Un autre attribut semble jouer un rôle dans l'attractivité des individus : la similarité. En effet, il semblerait que le fameux proverbe « Qui se ressemble s'assemble » serait fondé. Cette théorie soutenant que les être humains recherchent des personnes qui leur sont similaires est appelée l'homogamie.

Fiore et Donath [13] ont analysé des données provenant d'un site de rencontres en ligne concernant 8 mois d'activité entre Juin 2002 et Février 2003. Ils ont ainsi pu montrer que les utilisateurs concernés recherchaient fortement des individus qui leur étaient similaires à certains points de vue. Cette recherche de similarité concernait différentes caractéristiques. Le constat est évidemment plus frappant pour certaines que pour d'autres. Une caractéristique ayant une forte importance est le statut marital ou encore l'envie d'avoir des enfants. En effet, les valeurs concernant ces deux caractéristiques étaient les mêmes pour les dyades 64% et 54% plus souvent que lors d'alliances aléatoires. Le nombre d'enfants que l'utilisateur déclare déjà avoir avait également un impact important sur sa recherche. Dans une moindre

mesure, les adjectifs concernant le profil physique des individus et communs aux deux genres du types "average" ou "athletic" étaient liés.

Furman et Simon [14] se sont quant-à-eux intéressés à l'attitude des adolescents face à la recherche d'un partenaire. De nouveau, une homophilie statistiquement significative a pu être mise en évidence concernant le statut socio-économique ou même les résultats scolaires.

Kalmijn et Flap [25] sont allés un pas plus loin en s'intéressant aux causes de l'homophilie, en analysant l'impact que peuvent avoir le type d'endroit que les conjoints partageaient avant leur mariage (lieu de travail, école secondaire, quartier, connaissances familiales ou associations caritatives). Ils ont ainsi analysé quels contextes communs causaient le plus d'homophilie et de quel type (origines religieuses, origines socio-économiques, classe socio-économiques actuelles, éducation).

Nombreux sont les articles mentionnant le phénomène, nous pouvons terminer en citant également Buss [8] et Kalmijn [24].

Mais si le proverbe « Qui se ressemble s'assemble » semble dire vrai, qu'en-est-il d'autres célèbres dictons ? C'est la question que se posent Langlois et al.[29]. Les auteurs s'intéressent particulièrement à trois adages populaires :

- ▷ *tous les goûts sont dans la nature* [1],
- ▷ *ne jamais juger un livre à sa couverture* [2],
- ▷ *la beauté n'est qu'une qualité superficielle* [3].

Ils montrent alors que la sagesse populaire n'est pas toujours bien informée, ou du moins, qu'elle a ses limites. Dans leur étude, Langlois et al. parviennent à montrer l'existence d'un consensus conernant la beauté des individus. Ce consensus existe également entre ethnies et entre cultures. Les auteurs confirment également que s'il on nous répète de ne pas juger une personne sur son apparence, ce n'est pas pour autant chose faite. En effet, ils ont pu montrer que les enfants et les adultes attirants sont jugés et traités plus positivement, et cela même par des personnes familières. Enfin, ils

1. traduit de *Beauty is in the eye of the beholder*
2. traduit de *never judge a book by its cover*
3. traduit de *Beauty is only skin-deep*

indiquent également que les personnes séduisantes possèdent de plus grandes aptitudes sociales et une santé mentale légèrement meilleure.

Si l'apparence physique est un élément clé dans la recherche d'un partenaire [28], elle n'est pas tout, comme le montrent Regan et al. [35]. En effet, dans leur article s'intéressant aux caractéristiques que les hommes et les femmes recherchent chez leur partenaire pour des relations à court ou long terme, les auteurs ont constaté que les qualités dites internes (intelligence, gentillesse, ouverture d'esprit, ...) étaient généralement préférées aux qualités externes (beauté, statut social, ...), en particulier lors de la recherche d'un partenaire à long terme. Parmi les qualité externes se dessine une nouvelle hiérarchie. En effet, les qualités physiques semblent primer sur celles relatives au statut social et à la richesse des individus.

Chapitre 5

Un deuxième modèle pour les mariages : The Wedding Ring

Dans ce chapitre, nous décrivons le modèle proposé par Billari et al. dans leur article « The "Wedding-Ring" : An Agent-Based Marriage Model Based on Social Interaction »[7]. Dans l'élaboration de ce modèle, les auteurs supposent qu'une cause majeure provoquant les mariages est la pression sociale. Une personne sera plus ou moins désireuse de se marier selon la proportion de couples appartenant à son réseau social et en particulier au *groupe des personnes significatives* (*relevant others*).

Si chaque nouveau couple formé dans le réseau social d'un individu augmente son désir de se ranger, le modèle qu'ils décrivent se différencie d'un modèle épidémiologique car la pression sociale n'est pas un facteur suffisant pour provoquer un mariage. En effet, lorsqu'un individu est entouré de nombreux couples, son désir de se marier est peut-être très élevé mais le nombre de partenaires potentiels dans son réseau social est alors plus faible, rendant la recherche plus compliquée.

Dans leur modèle, Billari et al. ne supposent pas l'existence d'un réseau social au sens mathématique, ils considèrent la population comme appartenant à un espace dont ils réduisent la dimension à deux. Ces deux dimensions sont la localisation des individus et leur âge. La localisation est considérée comme une seule dimension car les auteurs supposent pour leur modèle que les agents vivent dans un monde qui suit une cercle. C'est d'ailleurs à cette hypothèse que le modèle doit son nom de « Wedding Ring ». La localisation

d'un individu est alors entièrement déterminée par son angle.

Le groupe des personnes significatives d'un individu $ind1$ est alors défini comme contenant un nombre aléatoire d'agents appartenant à un voisinage, dans le réseau social à deux dimensions, de cet individu. Cette définition est très bien illustrée par la figure 5.1, tirée de l'article [7], sur laquelle les deux dimensions du réseau sont représentées, l'âge correspondant à la dimension verticale.

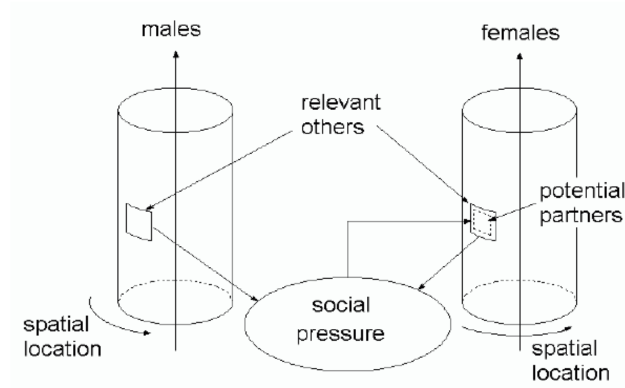

FIGURE 5.1 – Social network in the Wedding Ring – Tirée de [7]

Le réseau des personnes significatives d'un individu peut alors être représenté par deux intervalles. Le premier, symétrique, correspondant à la localisation des individus. Le deuxième est un intervalle autour de l'âge de l'individu et n'est pas nécessairement symétrique.

Les auteurs définissent alors la pression sociale, notée sp, en fonction du nombre de personnes mariées dans le réseau des personnes significatives de l'individu, noté pom, comme une fonction logit[40] :

$$sp = \frac{e^{\beta(pom-\alpha)}}{1 + e^{\beta(pom-\alpha)}}.$$

L'allure de la fonction est présentée à la figure 5.2

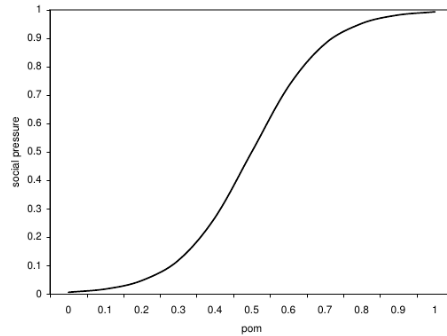

FIGURE 5.2 – Fonction de la pression sociale *sp* en fonction du nombre d'agents mariés dans le réseau des personnes significatives d'un individu *pom* – Tirée de [7] – $\beta = 7, \alpha : 0.5$

Le facteur de pression sociale *sp* sera utilisé pour définir un autre intervalle de dimension 2 qui correspondra à une « fourchette acceptable », un ensemble d'agents du sexe opposé que l'individu considère acceptables comme conjoints. Une pression sociale plus forte agrandira cet intervalle, ce qui correspond en fait à une diminution de l'exigence de l'individu. Cependant, la pression sociale ne sera pas seule à déterminer cette fourchette acceptable, l'âge a également une influence non négligeable. Les auteurs justifient cette décision par le fait que le réseau social d'un individu change de taille tout au long de sa vie, il serait d'ailleurs à son apogée entre 21 et 38 ans. L'influence de l'âge, *ai*, sur la fourchette acceptable est ainsi définie comme une fonction qui croît dans les plus jeunes âges, est à son maximum entre 21 et 38 ans et décroît ensuite en atteignant certains paliers. Cette fonction est représentée à la figure 5.3.

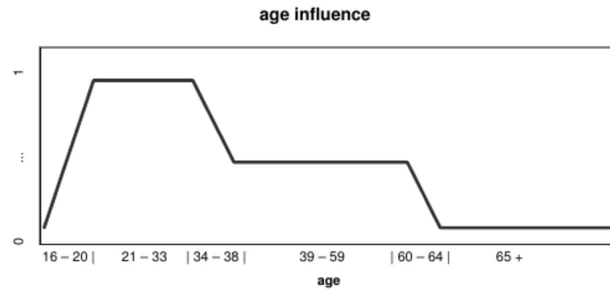

FIGURE 5.3 – Fonction de l'influence de l'age *ai* – Tirée de [7]

À partir de ces deux paramètres, ai et sp, les auteurs proposent alors de définir les intervalles correspondant aux partenaires potentiels.

La distance maximale autorisée pour la localisation de l'individu est donnée par

$$d = sp(pom) * m(N) * ai(x),$$

où x est l'âge de l'individu et N la taille de la population. Le facteur m, dépendant du nombre d'individus, est destiné à éviter que la probabilité de trouver un partenaire soit influencée par la taille de la population.

La différence d'âge maximale est définie comme

$$\Delta x = sp(pom) * ai(x) * c,$$

où c est une constante que les auteurs définissent à 25. Le produit $sp(pom) * ai(x)$ sera compris entre 0 et 1. Le paramètre c correspond donc à la plus grande différence d'âge possible.

Lorsque les partenaires potentiels ont été calculés, on étudie le cas de chacun d'entre eux. Lorsqu'un individu $ind2$ convient au nouveaux intervalles définis, on examine si $ind1$ convient également aux exigences de $ind2$. Si tel est le cas, un nouveau couple est formé.

Une fois le modèle spécifié, Billari et al. se consacrent à l'analyse des résultats de simulation. Ils y montrent la courbe de distribution des probabilités de se marier selon l'âge des individus qu'ils obtiennent. Cette courbe est qualitativement semblable à celle observée dans la réalité. Ils font ensuite évoluer certains de leurs paramètres afin d'en mettre l'intérêt en évidence. Ils réalisent ainsi des simulations en définissant une fonction de pression sociale linéaire, constante ou encore en définissant l'influence de l'âge comme constante.

Le modèle présenté est donc un modèle assez réaliste. En effet *The Weddind Ring* tient compte de deux critères : la désirabilité et la possibilité d'un mariage. La disponibilité est modélisée par l'ensemble des partenaires potentiels, par la dépendance de l'âge et le caractère mutuel de la décision de mariage. La désirabilité d'un mariage est, elle, représentée par la pression sociale et l'évolution de la taille du réseau social selon l'âge. Nous tenterons

de l'implémenter dans virtualBelgium et en analyserons les résultats dans le chapitre 8.

Chapitre 6

Analyse d'heuristiques de recherche séquentielle

6.1 Introduction

Dans le chapitre précédent, nous nous sommes intéressés à des méthodes d'affectation de partenaire dans le cadre de problème considérant une connaissance complète du problème. En effet, afin de réaliser nos mariages stables, chaque individu connaît l'ensemble de ses partenaires potentiels et peut donc les évaluer au préalable afin d'établir une liste de préférence.

Cependant, la vie nous soumet à une réalité bien différente. En effet, les partenaires potentiels vont et viennent, ils ne se présentent pas tous simultanément, attendant patiemment qu'un favori soit choisi. Dans ce chapitre, nous nous intéressons donc à des processus de recherche séquentielle, les candidats se présentant un à un et le choix étant donc fait sans forcément attendre d'avoir rencontrés tous les candidats.

La recherche d'un partenaire est donc ici considéré comme un problème de décision sous incertitude compliqué mais très intéressant. Selon les règles que nous décidons d'appliquer au problème, nous devrons faire face à différentes difficultés : l'incertitude concernant le nombre de prétendants, leur ordre l'apparition, la distribution de leurs valeurs, l'impossibilité de revenir en arrière, le coût de la recherche ou encore la nécessité de réciprocité.

Le principal problème à surmonter est le caractère mutuel de la recherche d'un partenaire. Chacun tente de se mettre en couple avec une personne ayant une grande valeur mais prend ainsi le risque d'être rejeté.

Nous commencerons en mentionnant différentes versions du problème de recherche séquentielle. Nous nous intéresserons à trois approches et verrons ainsi les conclusions tirées par des statisticiens, des économistes et des biologistes.

Ensuite, nous dresserons un rapide topo de la recherche unilatérale. Nous verrons l'efficacité de certaines méthodes très simples qui se révèleront particulièrement inadaptées lorsque le caractère mutuel est ajouté au problème.

Nous analyserons alors les heuristiques de recherche que proposent Todd et Miller[38] et recréerons leurs expérimentations. Nous pourrons ainsi aller un peu plus loin en modifiant certaines de leurs hypothèses. Nous nous attellerons donc également à comparer les résultats obtenus selon les hypothèses choisies.

6.2 Trois approches du problème

Dans leur article, Todd et Miller introduisent le sujet en décrivant trois approches de problèmes de recherche séquentielle. Bien que nous ne nous y attarderons pas, nous trouvons que cela constitue une très bonne introduction au sujet et aux différentes difficultés auxquelles nous devrons faire face dans notre recherche.

Un premier problème a été étudié par des statisticiens, il s'agit du *problème de la secrétaire*. Dans ce problème, un patron fait passer des interviews pour choisir une secrétaire, les candidates se succèdent dans un ordre aléatoire et la distribution de leurs qualité pour le job est inconnue. Cependant, le patron connaît le nombre exact de candidates qui se présenteront. Après chaque interview, il doit immédiatement décider s'il veut embaucher la candidate. Une fois partie, la candidate est perdue à jamais, aucun retour en arrière n'est autorisé. À la recherche de la perfection, seul le choix de la meilleure candidate est récompensé d'un profit de 1, le choix de toute autre

candidate, même excellente, se conclut par un profit de 0. Il a été montré que la meilleure stratégie pour ce problème est d'analyser sans jamais choisir les candidats d'un certain pourcentage du nombre total de postulants. On retient alors la meilleure candidate rencontrée pendant ces premières interviews, elle fixe ainsi notre seuil d'exigence. Après cette première phase, il faut embaucher la première candidate qui satisfait à notre exigence. De plus, il s'avère que la proportion optimale à utiliser pour la première phase du processus est de $1/e$, ce qui correspond à environ 37%. Ferguson[12] mentionne d'ailleurs qu'en suivant cette *règle des 37%*, on trouve la meilleure candidate dans 37% des cas, ce qui est assez impressionnant au vu de la complexité du problème.

Les économistes se sont plutôt intéressés à un problème légèrement différent concernant la recherche des prix les plus bas par les consommateurs. Contrairement au problème de la secrétaire, un consommateur a toujours l'opportunité de revenir dans un magasin qu'il a déjà visité, le processus de recherche a un coût et des solutions « moins que parfaites » sont acceptables. De plus, le nombre d'alternatives n'est pas connu. Dans de tels cas, le meilleur comportement à adopter pour un consommateur est de continuer à chercher un meilleur prix jusqu'à ce que l'effort de continuer la recherche soit probablement plus coûteux que la réduction de prix que l'on pourrait espérer, et ensuite retourner au meilleur prix rencontré jusqu'à présent. Cette méthode dépend donc fortement de l'écart-type de la distribution des prix, il peut donc être nécessaire de l'estimer à partir des prix déjà observés s'il est inconnu.

Du point de vue des biologistes, il est intéressant d'étudier le processus de recherche d'un partenaire. Différents modèles de comportements de recherche ont été établis au cours des années. Un effort particulier a été mené dans la comparaison des méthodes exigeant l'échantillonnage d'un certain pourcentage de la population avec des méthodes de type *Best-of-N* nécessitant l'échantillonnage d'un certain nombre N d'alternatives. Dans le premier type de méthode, on se retrouve dans des méthodes similaires à celle décrite pour le problème de la secrétaire, c'est-à-dire que l'on analyse un certain pourcentage de la population totale d'alternative afin d'établir un seuil d'exigence et on choisit ensuite la première alternative dépassant ce seuil. Pour les méthodes de type Best-of-N, N alternatives sont considérées et la meilleure d'entre elles est choisie, ce qui suppose donc un retour en arrière possible.

Dans le cadre de ces recherches sur le comportement humain, Simon introduit le terme « satisficing »[41]. Ce terme, formé à partir des mots *satifying* et *sufficing* pouvant être traduits comme satisfaisants et suffisants, traite de la capacité à accepter des solutions « moins-que-parfaites ». On s'écarte d'une attitude optimisatrice en établissant à la place un seuil d'exigence, ou seuil de satisfaction, introduisant ainsi des alternatives *suffisamment bonnes* bien que non optimales. Bien que ce terme n'ait pas encore d'équivalent fixé par l'usage[2], la banque de données terminologiques et linguistiques du gouvernement du Canada proposent la traduction de « rationalité de la suffisance ». Cette traduction nous semble très intéressante puisque « *Simon voit satisficing comme une des formes principales de la rationalité limitée disponible dans des situations dans lesquelles l'ensemble complet des alternatives est inconnu* » [38].

Remarquons que malgré les différences entre ces quelques approches, qu'elles autorisent le retour en arrière ou l'interdisent, que le choix doive être optimal à tout prix ou qu'un choix puisse être considéré suffisamment bon sans être le meilleur, ou encore que certains paramètres du problème comme le nombre d'alternatives ou la distribution qu'elles suivent soient supposés connus ou non, toutes traitent un problème en sens unique. Un sujet doit trouver une meilleure ou suffisamment bonne alternative sans faire face au risque de rejet de cette dernière. Le choix ne doit pas être mutuel. Or, une difficulté majeure de la recherche d'un partenaire est que l'intérêt doit être partagé. Cette subtilité complexifie drastiquement la recherche de méthodes efficaces de détermination du caractère satisfaisant d'une alternative.

6.3 Recherche unilatérale

Dans cette section, nous considérons le problème de recherche d'un partenaire dans le cas unilatéral, nous n'envisageons donc pas la possibilité de refus du candidat choisi. Cependant, comme dans le problème de la secrétaire, nous supposons le retour en arrière interdit. Cette hypothèse nous semble raisonnable lorsqu'il est question de recherche d'un partenaire. En effet, bien que possible en réalité, il est assez mal vu de revenir sur sa décision auprès de quelqu'un qu'on aurait rejeté à défaut d'avoir trouvé de meilleur candidat.

Même sans ajouter le risque de refus de l'autre parti, le problème révèle déjà certains défis dus à l'ignorance concernant les futurs partenaires potentiels. La distribution de leurs *valeurs* est inconnue, leur nombre l'est également. Comment alors décider se lier à une personne sans savoir si une autre, plus intéressante encore, pourrait se présenter par la suite ? De même, comment prendre une décision sans même connaître le nombre de partenaires potentiels nous pourrions rencontrer si nous continuions notre recherche ?

Nous avons vu lors du problème de la secrétaire qu'une excellente stratégie est la règle des 37%. Cependant, dans ce problème, le nombre de candidats était connu, ce qui n'est plus le cas ici. De plus, si le nombre de partenaires potentiels est important, une telle méthode exige l'analyse d'un très grand nombre de candidats avant de prendre la moindre décision, ce qui ne nous semble pas très cohérent avec notre problème de recherche d'un partenaire amoureux.

Pour analyser les résultats d'une telle méthode, Todd et Miller réalisent différentes simulations. Lors d'une première simulation, ils supposent une population de 100 partenaires potentiels suivant une distribution uniforme et étudient la chance de se lier à un partenaire d'une certaine catégorie selon le nombre d'individus considérés dans une première phase. On se retrouve donc avec des méthodes de recherches similaires à celle des 37%, on évalue d'abord un certain nombre de candidats sans jamais les choisir afin d'établir un seuil de satisfaction que devra dépasser un partenaire potentiel afin d'être choisi. Dans cette analyse, l'optimalité n'est pas obligatoire, il ne faut donc pas se contenter d'évaluer la chance de choisir le meilleur candidat. Il est, par exemple, intéressant de considérer la chance de finir avec un individu faisant partie du haut du panier, que nous définirons ici comme les 10% ou 25% de meilleurs candidats. De plus, la chance de se retrouvé avec l'un des pires candidats, appartenant aux 25% des plus valeurs les plus basses, est également évaluée.

Nous avons réalisé des simulations [1] et avons obtenus des graphes similaires à ceux obtenus par Todd et Miller. Notons que les rencontres suivant un ordre aléatoire, nous avons réalisé plusieurs simulations et ensuite pris les

1. Les codes utilisés pour réaliser les figures 6.1 et 6.2 ont été fournis en annexe B

moyennes de nos différents résultats. Dans le cas de la recherche unilatérale, nous avons effectués 10 000 simulations d'un individu à la recherche d'une partenaire parmi N. La figure 6.1 présente nos résultats pour 100 partenaires potentiels.

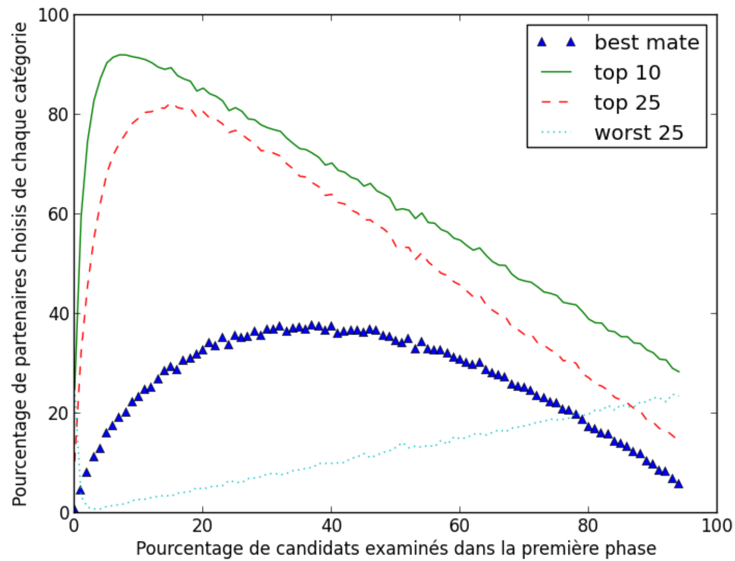

FIGURE 6.1 – Chance de trouver un partenaire d'une certaine tranche de la population en fonction du nombre de candidats évalués avant la détermination d'un seuil d'exigence - $N = 100$ - Distribution uniforme

Précisons notre légende :

- ▷ *Best mate* correspond à la chance d'avoir sélectionné le meilleur candidat de tous ;
- ▷ *Top 10* correspond à la chance d'avoir sélectionné un candidat faisant partie des 10% des meilleurs candidats ;
- ▷ *Top 25* correspond à la chance d'avoir sélectionné un candidat faisant partie des 25% des meilleurs candidats ;
- ▷ *Bottom 25* correspond à la chance d'avoir sélectionné un candidat faisant partie des 25% des pires candidats.

Remarquons que lors de nos simulations, si l'individu ne rencontre pas de candidat satisfaisant ses attentes, il est alors mis en couple avec le dernier partenaire potentiel se présentant, c'est ce qui explique la croissance de la courbe correspondant aux 25% des pires candidats quand le nombre de candidats évalués dans la phase d'ajustement du seuil de satisfaction augmente.

Nous pouvons également remarquer que l'affirmation selon laquelle la méthode des 37% est notre meilleure chance de trouver le meilleur partenaire se confirme.

Une deuxième simulation est alors réalisée lorsque le nombre de candidats est plus important : 1000. À nouveau, nos résultats sont similaires à ceux obtenus par Todd et Miller et nous les présentons à la figure 6.2.

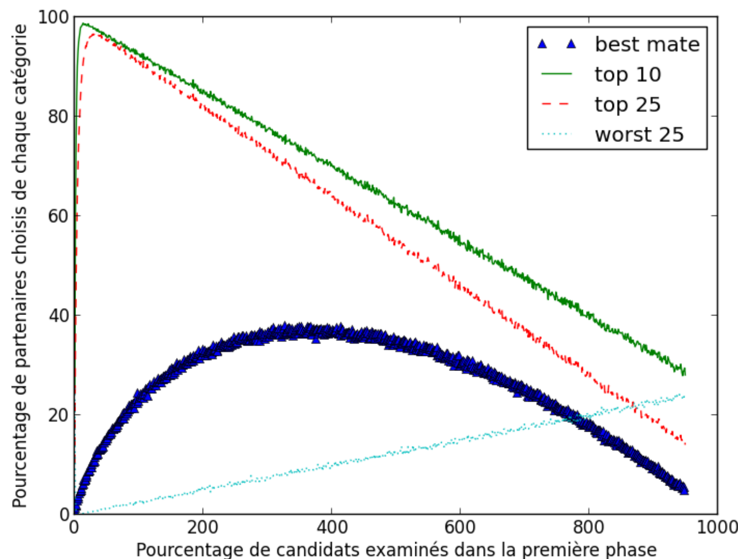

FIGURE 6.2 – Chance de trouver un partenaire d'une certaine tranche de la population en fonction du nombre de candidats évalués avant la détermination d'un seuil d'exigence - $N = 1000$ - Distribution uniforme

Nous remarquons que le comportement obtenu avec 1000 candidats ressemble à celui obtenu avec seulement 100 partenaires potentiels. La règle des 37% est toujours optimale si l'unique but est de trouver le meilleur partenaire.

Cependant, avec une population plus grande, la croissance dans les premiers pourcentages est bien plus forte. Cela signifierait donc que si, contrairement au problème de la secrétaire, l'optimalité n'est pas exigée, alors il n'est peut-être pas réellement nécessaire d'évaluer un certain pourcentage de la population mais simplement un certain nombre de candidats. Une heuristique de type *Try a Dozen* (Essaie une douzaine) semble plus appropriée à un tel problème. Pour tenter d'avoir une idée plus précise du nombre d'individus qu'il serait intéressants d'évaluer, nous avons réalisé les mêmes graphiques mais en limitant notre analyse à 50 individus rencontrés lors de la phase d'ajustement. Les résultats obtenus sont présentés à la figure 6.3. Nous pouvons y voir que pour 100 partenaires potentiels, il est intéressant d'en évaluer 10, pour 1000 candidats, il semble approprié d'en évaluer entre 10 et 30 selon l'exigence (selon qu'on veuille maximiser la chance de choisir un candidat de type *Top 10* ou *Top 25*).

Figure 6.3 – Chance de trouver un partenaire d'une certaine tranche de la population en fonction du nombre de candidats évalués avant la détermination d'un seuil d'exigence - Comparaison $N = 100$ et $N = 1000$ - Distribution uniforme

Intuitivement, nous pensons que la distribution des valeurs des candidats devrait plutôt se rapprocher d'une loi normale. Nous avons donc décidé de recommencer nos simulations mais cette fois avec des partenaires potentiels suivant des distributions uniformes. Plusieurs valeur d'écart-type ont été testés afin d'en étudier l'influence. Remarquons que pour ces graphiques, nous n'avons pas pu comparer nos résultats à ceux de Todd et Miller qui ont supposé l'uniformité de la distribution tout au long de leur article.

6.4 Recherche bilatérale

Repartons désormais de notre heuristique *Try a Dozen* obtenue lors de notre analyse du cas unilatéral. Supposons une population composée de 100 hommes et 100 femmes dont les valeurs sont distribuées suivant une loi uniforme $U([0,1])$. Supposons que tous nos individus suivent cette méthode de recherche. On imagine bien qu'une telle situation n'apportera du bonheur qu'aux individus appartenant au haut du panier. En effet, nous avons vu

dans la section précédente qu'en déterminant notre seuil d'exigence de la sorte, nos chance d'obtenir un partenaire faisant partie des meilleurs 10% étaient très élevées. Cependant, une offre doit désormais être acceptée, le partenaire potentiel satisfaisant aux exigences d'un individu l'évaluera en retour et n'acceptera son offre que s'il est jugé suffisamment bon. Seuls les meilleurs candidats, hommes ou femmes, finiront donc en couple et la majorité de notre population restera célibataire.

Il suffit peut-être alors d'être moins exigeant et donc de réduire le nombre d'individus examinés lors de la phase d'ajustement. Le seuil d'exigence sera alors moins élevés et plus de couples se formeront. Mais est-ce réellement une solution viable ?

Dans cette section, à nouveau, nous réalisons des simulations numériques afin d'analyser l'évolution du système selon certains critères [2]. Nous pourrons ainsi comparer différentes heuristiques proposées, en évaluer la qualité dans notre type de problème et mieux comprendre leurs conséquences. À nouveau, pour contrer les problèmes dû à l'aléatoire, nous avons réalisé plusieurs simulations, ici 100, et ensuite pris les moyennes de nos différents résultats. Pour nos simulations, nous diviserons l'évolution en deux phases. La première phase correspond à la phase d'ajustement de notre section précédente. Nous appellerons cette phase l'adolescence. Durant cette dernière, les individus rencontrerons différents individus qui influenceront leur seuil d'exigence mais également qui seront retirés de leurs partenaires potentiels. En effet, nous supposons à nouveau que les retours à un candidat déjà rencontrés sont interdits. Ensuite, la seconde phase sera commune à toutes les méthodes puisqu'il s'agira simplement de faire des rencontres aléatoires et de mettre en couple deux personnes dès que les deux se conviennent mutuellement. La simulation s'arrête une dois que l'on ne peut plus trouver deux individus célibataires qui ne se sont encore jamais rencontrés.

L'influence que chaque individu aura sur le seuil d'exigence de la personne rencontré dépendra de la méthode analysée. Nous analyserons alors, pour chaque méthode définie, le nombre de couples formés lors de la simulation, la différence moyenne entre partenaires et la valeur moyenne des personnes

2. Les codes utilisés pour réaliser les figures de cette section ont été fournis en annexe C

ayant trouvé l'amour en fonction de la longueur de l'adolescence.

Nous pouvons alors définir une première méthode pour nous retrouver avec une heuristique du type *Try A Dozen*. Nous appelerons la méthode *Take Next Best* (TNB). Lors de l'adolescence, le seuil d'exigence est déterminé comme dans notre précédente section. À chaque individu rencontré, on détermine le seuil comme le maximum entre le seuil précédent et la valeur du partenaire potentiel. Nous aurons donc la méthode *Try A Dozen* pour une adolescence correspondant à 12 rencontres. Nos prédictions se confirment suite aux simulations. En effet, la figure 6.4 indique que le nombre de couples formés est déjà très faible pour une adolescence correspondant à 12 rencontres et diminue au plus dure l'adolescence. On remarque également à la figure 6.5 que la valeur moyenne des individus ayant trouvé un partenaire est extrêmement élevée. De plus, nous avons, puisque seuls les individus très bien classés se mettent en couple, que la différence moyenne entre partenaires est très faible comme le suggère la figure 6.6. Remarquons que s'il on diminue la longueur de l'adolescence, en la limitant par exemple à une seule rencontre, nous avons un pourcentage acceptable de formation de couple mais la différence moyenne au sein des couples explose, ce qui ne semble pas réaliste et donc pas acceptable.

Le problème est que chacun définit son seuil d'exigence sans tenir compte de sa propre valeur. Ainsi, le pire candidat aura environ les mêmes exigences que le meilleur. Il semble pourtant réaliste d'ajuster ses attentes à sa propre valeur. Nous pouvons alors définir une nouvelle heuristique de recherche, que nous appellerons *humble*, adaptant les exigences de chacun à sa propre valeur. Avec cette méthode, nous fixerons le seuil d'exigence d'un individu comme sa propre valeur diminuée de 5. En suivant cette méthode, nous obtenons une proportion de mise en couple très importante comme l'indique la figure 6.4. De plus, la différence moyenne entre partenaires reste raisonnable et la valeur moyenne des individus ayant trouvé un partenaire est proche de la valeur moyenne de l'ensemble de la population. Remarquons qu'avec cette méthode, le seuil d'exigence est fixé est n'est donc pas modifié au cours de l'adolescence, sa longueur n'aura donc que peu d'influence si ce n'est que plus cette phase dure, moins de partenaires potentiels pourront se présenter à l'individu lors de la deuxième phase.

Cette méthode semble donc très intéressante, cependant que penser de

l'hypothèse selon laquelle un individu connaisse avec exactitude sa valeur propre ? Il est évident que l'on ne se voit jamais comme un partenaire potentiel nous voit. Il est très compliqué de porter un jugement objectif à notre propre sujet. Il nous faut alors évaluer notre valeur d'une certaine manière.

Une première méthode est d'élever le seuil d'exigence d'un individu, correspondant à la perception qu'il a de sa propre valeur, à chaque proposition qu'il reçoit. On diminue alors son seuil d'exigence à chaque refus encaissé. On commencera alors avec un seuil d'exigence initialisé à 50 pour tous les individus et on définira un ajustement inversement proportionnel à la longueur de l'adolescence. On obtient alors la méthode que nous appellerons *adjust1* qui marie environ 40% de la population mais préférablement les individus de la moitié basse comme le montre la figure 6.5. Le problème de cette méthode est qu'elle ne considère pas la valeur des partenaires potentiels lors de leurs offres ou de leurs refus. Pourtant nous ne devrions pas être flattés par une offre venant d'un individu que nous considérons comme « inférieur », une telle offre ne devrait doc pas augmenter notre perception de nous mais simplement nous sembler logique. Avec cette méthode, les individus ayant une valeur supérieure à 50 vont donc recevoir de nombreuses offres et augmenter leurs attentes à chacune d'entre elles atteignant ainsi un seuil d'exigence trop élevé. Les personnes dont la valeur se situe en dessous de 50 seront quant-à-elles souvent rejetées et diminuer leurs attentes, continuant ainsi à flatter la moitié supérieure de la population. Cependant, les individus dont les attentes ont été fortement réduites trouveront plus facilement un conjoint que les individus trop sûrs d'eux et donc trop exigeants.

Pour pallier à ce problème, Todd et Miller proposent de prendre en compte la valeur des partenaires potentiels avant d'opérer quelque ajustement que ce soit sur la perception d'un individu de sa valeur propre. On n'ajuste alors cette perception que lorsque l'offre ou le rejet paraît surprenant étant donné la perception actuelle de l'individu. Cela signifie que l'on ne modifie la valeur perçue d'un individu que lorsqu'il reçoit une offre d'un partenaire potentiel ayant une valeur supérieure à sa propre perception ou lorsqu'il essuie un refus d'un candidat de valeur inférieure à sa propre perception. Dans les autres cas, rien ne semble étonnant et il n'y a donc aucune raison d'augmenter ou diminuer ses attentes. Nous noterons cette méthode *adjust2*. Nous obtenons à peu près la même proportion de formation de couples qu'avec la méthode précédente mais c'est désormais la moitié supérieure de la population qui a

plus de facilité à trouver un partenaire étant donné que la valeur moyenne des individus en couple tourne aux alentours de 75 (figure 6.5). Cependant, les couples formés ne sont plus très équilibrés comme nous pouvons le constater en observant la figure 6.6. En effet, c'est cette méthode qui crée les couples les plus disparates.

De nouveau, le problème vient du fait qu'on ne tient pas suffisamment compte de la valeur des partenaires potentiels. En effet, s'il on s'intéresse à la valeur d'un individu afin de savoir si son offre ou son refus doit provoquer une remise en question, l'ampleur de l'ajustement n'en dépend pas. Une dernière méthode proposée par Todd et Miller est donc de considérer la valeur du partenaire potentiel pour la détermination de l'ajustement à réaliser. L'ajustement est alors défini comme la moitié de la différence entre la valeur du partenaire potentiel et la valeur perçue de l'individu. Cet ajustement n'est réalisé que lorsqu'il est nécessaire comme expliqué précédemment pour la méthode *adjust2*. Avec cette dernière méthode, *adjust3*, nous obtenons des résultats plus proches de la méthode *humble* qui nous semblait très intéressante mais peu réaliste.

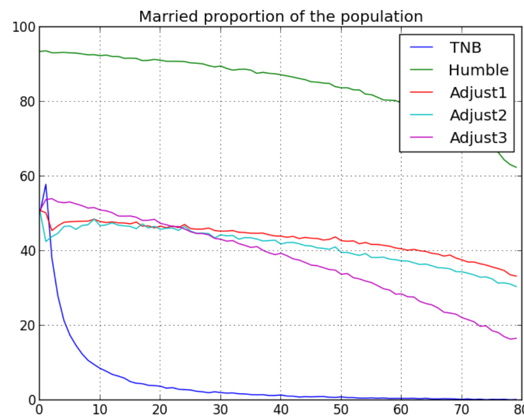

FIGURE 6.4 – Nombre de couples formés lors d'une recherche séquentielle bilatérale et selon la méthode utilisée - Distribution uniforme

L'hypothèse d'une distribution uniforme dans la population nous semblant peu réaliste, nous avons voulu savoir si les résultats changent fortement si les valeurs des individus suivent une distribution normale. Nous avons

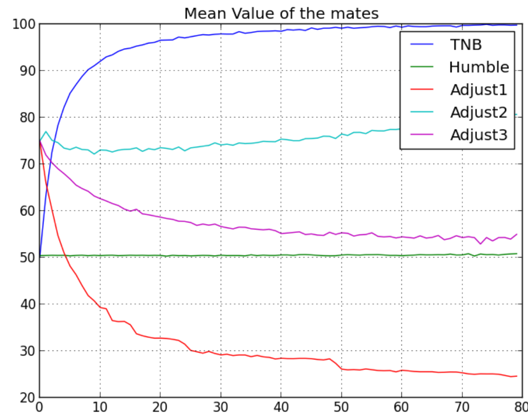

FIGURE 6.5 – Valeur moyenne des individus ayant trouvé un partenaire lors d'une recherche séquentielle bilatérale et selon la méthode utilisée - Distribution uniforme

donc recréés les même graphes avec cette fois-ci, une distribution normale N(50,10). Les résultats sont présentés aux figures 6.7, 6.8 et 6.9.

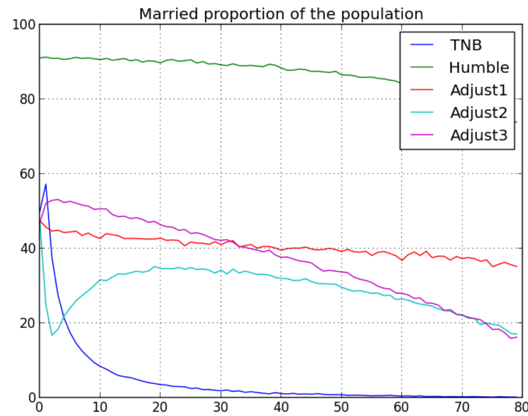

FIGURE 6.7 – Nombre de couples formés lors d'une recherche séquentielle bilatérale et selon la méthode utilisée - Distribution normale N(50,10)

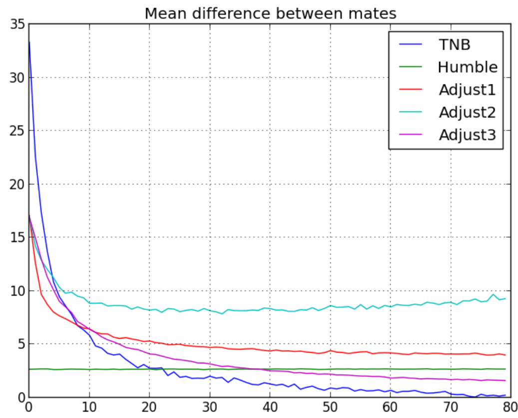

FIGURE 6.6 – Différence moyenne de valeur au sein des couples formés lors d'une recherche séquentielle bilatérale et selon la méthode utilisée - Distribution uniforme

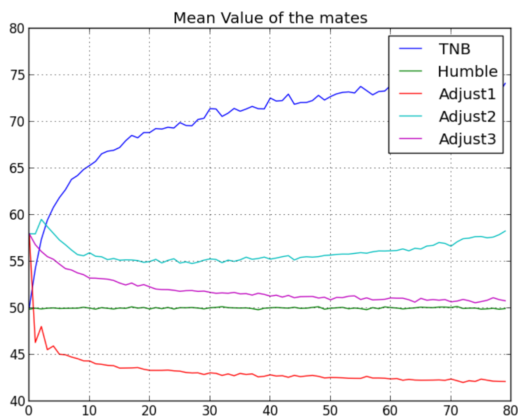

FIGURE 6.8 – Valeur moyenne des individus ayant trouvé un partenaire lors d'une recherche séquentielle bilatérale et selon la méthode utilisée - Distribution normale N(50,10)

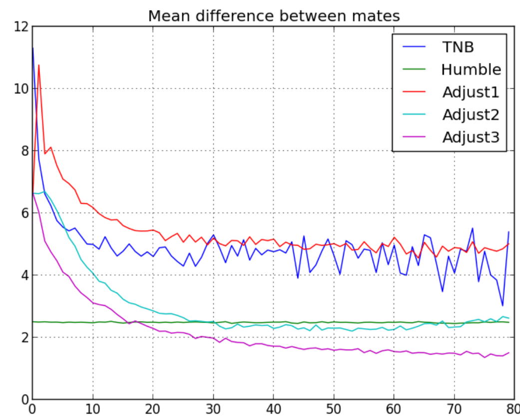

FIGURE 6.9 – Différence moyenne de valeur au sein des couples formés lors d'une recherche séquentielle bilatérale et selon la méthode utilisée - Distribution normale N(50,10)

Chapitre 7

Principes du logiciel Repast HPC

Pour comprendre le fonctionnement de VirtualBelgium, il nous a fallu investiguer le fonctionnement de l'outil Repast HPC. C'est pourquoi nous tentons de résumer son manuel [10] dans ce chapitre.

Repast est un outil pour la simulation et modélisation de systèmes multiagents. Dans le cadre de VirtualBelgium, projet nécessitant beaucoup de puissance de calculs, c'est la version Repast for High Performance Computing (Repast HPC) qui a été utilisée.
Repast HPC est écrit en C++, il utilise MPI pour les opérations parallèles ainsi que la librairie boost [1].

Les agents sont implémentés comme des classes C++. L'état d'un agent est représenté par les attributs de la classe correspondante et son comportement par ses méthodes.

Pour la simulation, Repast HPC fonctionne selon un principe de planning. À chaque itération de la simulation, l'événement suivant du planning se présente et est exécuté.
L'utilisateur planifie un événement à un certain « tick » et les ticks déterminent l'ordre des événements.

1. Boost.MPI [18] est une libraire de programmation parallèle basée sur la librairie standard MPI (Message Passing Interface), librairie la plus populaire pour le calcul distribué de haute performance, mais rendue plus accessible en C++.

7.1 Simulation parallèle

Agents parallèles

Repast HPC est conçu pour un environnement parallèle dans lequel de nombreux processeurs tournent et dont la mémoire n'est pas partagée. Les agents sont distribués sur les différents processeurs : à chaque agent correspond un seul processeur responsable d'exécuter le code de son comportement. On dira alors que l'agent est *local* au processeur.

Évidemment, un processeur peut avoir besoin d'informations sur des agents qui ne lui sont pas locaux, Repast permet alors de partager des *copies* des agents parmi les processeurs et permet donc aux agents locaux d'interagir avec des agents non-locaux.

Pour pouvoir distinguer les différents agents, chacun possède un id unique représenté par la classe `AgentId`. Cette classe possède quatre composantes :

- id : un id numérique spécifié par l'utilisateur (cet id devrait être différent pour tous les agents créés sur un même processeur)
- agent type : un entier correspondant à la classe C++ de l'agent
- starting rank : le rang du processeur sur lequel l'agent a été créé
- current rank : le rang du processeur auquel l'agent est local actuellement.

Toutes les classes agents doivent implémenter la classe abstraite[2] `Agent` qui fournit l'`AgentId` de l'agent.

Communication entre processeurs et synchronisation

Puisque les simulations de Repast HPC sont réparties sur plusieurs processeurs, il est souvent nécessaire de communiquer ou synchroniser les informations parmi les processeurs.
Cette nécessité apparaît dans les cas suivants :

- lorsqu'un processeur a besoin de copies d'agents d'un autre processeur,
- lorsqu'un processeur possède des copies d'agents non-locaux qui doivent être mises à jour,

2. Vous trouverez en annexe quelques détails sur les classes abstraites

- lorsqu'un agent doit être complètement déplacé (et non juste copié) d'un processeur à un autre.

La communication entre processeurs est fortement automatisée par Repast HPC. Cependant, l'utilisateur doit fournir le code de *serialization*[3] du type, extraire l'état de l'agent et l'*emballer* pour le transfert puis le *déballer* et créer ou mettre à jour l'agent approprié à partir du *paquet* reçu.

Repast HPC appelle cette méthode d'emballage et déballage le « modèle Package ».
L'utilisateur doit alors fournir un Package contenant

- l'état à communiquer à un autre processeur,
- un *Provider* qui fournit un Package à partir d'un agent donné,
- un *Receiver* qui reçoit le Package, le « déballe » et crée ou met à jour un agent à partir du Package.

7.2 Écrire un modèle Repast HPC

Une simulation Repast HPC typique consiste en

- un certain nombre de classes Agent,
- le code Package nécessaire,
- une classe « model »,
- une fonction `main`.

Nous allons détailler ces éléments.

7.2.1 Classes Agent et code Package

Comme dit précédemment, tous les agents doivent implémenter la classe abstraite `Agent` dont l'interface est la suivante.

```
1 class Agent {
2 public :
3     virtual ~Agent() {}
```

3. L'opération de « Serialization » de la librairie boost [34] désigne la déconstruction réversible d'un ensemble de structures de données C++ en une séquence de bytes, appelée alors *archive*.

```
    virtual AgentId& getId() = 0;
    virtual const AgentId& getId() const = 0;
};
```

Prenons un exemple très simple d'un agent n'ayant pour état qu'une seule variable entière `_state`.

```
class ModelAgent: public repast::Agent {

private:
    repast::AgentId _id;
    int _state;

public:

    ModelAgent(repast::AgentId id, int state);
    virtual ~ModelAgent();

    int state() const {
       return _state;
    }

    void state(int val) {
       _state = val;
    }

    repast::AgentId& getId() {
        return _id;
    }

    const repast::AgentId& getId() const {
        return _id;
    }

   void flipState();

};
```

Les packages sont typiquement implémentés comme des *struct* contenant le minimum nécessaire pour décrire l'état d'un agent. Nous pouvons implémenter le Package pour notre exemple.

```
struct ModelAgentPackage {

    friend class boost::serialization::access;
    template<class Archive>
    void serialize(Archive& ar, const unsigned int version
        ) {
        ar & id;
        ar & state;
    }

    repast::AgentId id;
    int state;

    repast::AgentId getId() const {
        return id;
    }
};
```

7.2.2 La classe « Model »

La classe `Model` n'implémente aucune interface spécifique ou n'étend aucune classe.
Une classe Model typique est principalement responsable de

- créer un *SharedContext* et le remplir d'agents
- Planifier la simulation d'actions
- Effectuer une synchronisation initiale des agents et projections

Un SharedContext contient tous les agents locaux et non-locaux actuellement sur un processeur et implémente un certain nombre de méthodes pour en ajouter ou en retirer ainsi que des itérateurs pour boucler sur les agents locaux ou non-locaux.

```
class Model {

private:
    int rank;

public:
```

```
7      repast::SharedContext<ModelAgent> agents;
8      repast::DataSet* dataSet;
9
10     Model();
11     ~Model();
12     void initSchedule();
13     void step();
14 }
```

7.2.3 La fonction `main`

La fonction `main` s'occupe

- des initialisations de Repast HPC et MPI,
- de créer la classe Model et d'y appeler toutes les initialisations nécessaires,
- de lancer le ScheduleRunner
- d'appeler d'éventuels codes de nettoyage.

Chapitre 8

Implémentation

8.1 Introduction

Dans ce chapitre, nous tentons de décrire nos implémentations des différents modèles analysés. Dans un premier temps, nous décrivons les méthodes de base utilisées par tous les modèles. Ces méthodes correspondent à l'implémentation d'un mariage, d'un divorce et à la définition du potentiel attractif d'un individu. Ensuite, nous expliquons quatre modèles tour à tour, nous en résumons l'implémentation et détaillons les différents problèmes rencontrés. Enfin, nous terminons par une comparaison des trois modèles et de leurs résultats.

8.2 Méthodes communes aux trois modèles

8.2.1 Méthode simulant un mariage

Pour simuler un mariage, nous avons créé la méthode `mergeHousehold`. Cette méthode prend en argument les deux individus désirant se marier *ind*1 et *ind*2. Tous les individus du ménage de *ind*2 sont alors ajoutés à celui de *ind*1 et le type du ménage est recalculé. Il faut également penser à modifier différentes caractéristiques des individus qui ont attrait à leur ménage : leur maison, leur code ins, l'id de leur ménage ainsi que leur statut au sein de celui-ci[1]. Enfin, le ménage de *ind*2 est supprimé.

1. Notons que le modèle suppose donc lors d'un mariage que l'individu *ind*2 emménage chez l'individu *ind*1, un modèle concernant les déménagement étant le sujet de mémoire

8.2.2 Méthode simulant un divorce

Pour réaliser le divorce de deux individus, nous avons créé la méthode `divorce` qui prend en arguments l'id du ménage à scinder ainsi les deux partenaires : le chef du ménage, *head*, et son conjoint, *mate*. L'opération réalisée est très simplifiée puisque l'on crée simplement un nouveau ménage uniquement composé de *mate* que l'on retire alors du ménage initial. Un noeud du réseau est alors tiré aléatoirement afin de définir la nouvelle demeure de *mate* tandis que *head* et le reste de la famille ne changent rien à leurs habitudes. Le problème de la garde d'enfants ou même d'adultes supplémentaires n'a donc pas été envisagé dans le cadre de ce mémoire mais il serait intéressant de s'y attarder quelque peu. Dans notre contexte, nous avons supposé que le chef de famille restait responsable de l'entièreté de son ménage et que le seul changement était le départ du conjoint.

8.2.3 Définition du potentiel attractif des individus

Afin de définir le potentiel attractif d'un individu, nous nous sommes basés sur la théorie de l'homogamie présentée à la section 4.4. Un individu $ind1$ trouvera alors un autre individu $ind2$ plus attractif au plus ce dernier lui ressemble. Dans nos attributs décrivant les individus et susceptibles de déterminer une similarité entre agents, nous comptons l'âge, le niveau d'éducation, le nombre de mariages déjà vécus ainsi que le statut socio-professionnel.

L'importance que prenne chacun de ces paramètre a été définie intuitivement mais une étude plus poussée ou, du moins, une analyse de leur impact serait la bienvenue.

Nous avons trouvé plus simple de donner une mesure de dissimilarité, cette dernière est calculée à l'aide de notre méthode `compute_dissimilarity`. Cette dernière tient compte des différents paramètres de la façon suivante.

- ▷ La différence d'âge est comptabilisée par tranche de 3 ans : on ajoute à notre mesure de dissimilarité le résultat de la division entière de la différence d'âge par 3.

d'E. Ramelot, il serait intéressant, par la suite, de conjuguer les deux modèles afin de déterminer si le nouveau couple déménage.

▷ Pour calculer la différence du niveau d'éducation, nous attribuons un entier à chaque type d'éducation proposé dans VirtualBelgium. Nous aurons alors 0 pour une personne sans diplôme, 1 pour une personne possédant uniquement un diplôme d'école primaire, 2 pour un individu ayant un diplôme secondaire et 3 pour un diplôme universitaire ou de haute-école. Nous pouvons dès lors soustraire les deux entiers obtenus. Nous ajoutons alors à notre dissimilarité le carré de cette soustraction.

▷ En ce qui concerne le nombre de mariages de chaque individu, nous en gardons la différence en valeur absolue et la multiplions par 2, ce que nous ajoutons alors à notre dissimilarité.

▷ Enfin, nous ajoutons une valeur de 3 à notre dissimilarité si les deux individus n'ont pas le même statut socio-professionnel. Rappelons que ce statut dans VirtualBelgium peut prendre trois valeurs : actif, inactif ou étudiant.

8.3 Description des modèles

8.3.1 Premier modèle : mariages et divorces

Le premier modèle que nous avons implémenté dans VirtualBelgium combine le modèle décrit au chapitre 3 à l'algorithme de Gale-Shapley discuté dans le chapitre 4. Ce modèle, contrairement aux suivants, combine la modélisation des mariages et celle des divorces.

Dans un premier temps, il faut implémenter la méthode de décision concernant le choix de se marier ou non. Nous avons donc ajouté une nouvelle méthode à la classe `Individual` : `marital_choice`. Cette méthode renverra True si l'individu décide de se marier (ou de rester marié) et False dans le cas contraire. Cette méthode implémente simplement le modèle présenté dans la section 3.2.

Une fois la méthode de décision appliquée, il faut l'appliquer à chaque individu concerné. Deux types d'individus sont à distinguer car leurs traitements diffèrent. En effet, nous devons séparer le cas des agents qu'il faudra faire divorcer du cas des agents à marier.

D'une part se trouvent les individus en couple, ils peuvent être identifiés à l'aide du type de leur ménage qui correspond soit à une famille, 'F', soit à un couple sans enfant, 'C'. Ces individus sont ceux qui sont susceptibles de divorcer. La méthode `marital_choice` est appelée sur les deux partenaires : le chef de famille et son conjoint. Si l'un d'entre eux ne désire plus être marié, le processus de divorce est lancé.

D'autre part, nous retrouvons les individus célibataires, divorcés ou veufs. Ces derniers peuvent vouloir se marier ou se remarier. Nous définissons alors deux vecteurs `list_single_men` et `list_single_women` auxquels nous ajoutons les individus désirant trouver un époux. À partir des ces deux listes de célibataires à la recherche d'un conjoint, nous pouvons créer des couples. C'est ici qu'entre en jeu le problème des mariages stables. En effet, afin de créer des couples suffisamment réalistes, nous ne pouvons pas nous contenter de marier l'homme en numéro i avec la femme en numéro i.
Pour nous ramener au problème des mariages stables décrit au chapitre 4, nous devons établir les préférences de chaque célibataire de nos listes. Pour cela, nous nous sommes basés sur la théorie de l'homogamie comme mentionné précédemment. Nous avons ainsi défini une méthode définissant la dissimilarité existant entre deux individus et ensuite simplement défini l'ordre de préférence comme suivant l'ordre décroissant des dissimilarités ainsi définies. Nous pouvons ensuite appliquer l'algorithme de Gale-Shapley et obtenir un couplage stable. Une fois le couplage stable défini, il suffit de marier les couples créés.

Cependant, nous devons garder à l'esprit le problème que nous traitons. En effet, VirtualBelgium implémente une population synthétique initiale de de 10 262 160 individus. Même si les individus célibataires, à la recherche d'un partenaire, et en âge de se marier, ne représentent qu'une fraction d'entre eux, ils restent extrêmement nombreux. Le simple calcul des listes de préférences devient alors beaucoup trop gourmand en mémoire et cela sans parler de l'algorithme de Gale-Shapley d'une complexité quadratique qui risque d'avoir un impact trop important sur le temps d'exécution du programme. Nous avons donc décidé de diviser le problème en de nombreux sous-problèmes. Pour cela, nous découpons nos vecteurs contenant les célibataires en plus petits vecteurs sur lesquels appliquer tour à tour l'algorithme de Gale-Shapley. Ces plus petits vecteurs ont une taille définie à 50 excepté le dernier, contenant $N \bmod 50$ individus où N est le nombre d'individus du plus petit ensemble

de célibataires (hommes ou femmes).

Cette découpe en sous-problèmes s'avère très efficace en ce qui concerne le temps de calcul [2].

8.3.2 Deuxième modèle : mariages

Pour ce second modèle, nous nous sommes inspirés du modèle *The Wedding Ring* présenté au chapitre 5 que nous avons légèrement adapté à notre situation. Dans ce cas, seuls les mariages sont simulés. Il faut donc combiner ce modèle à un modèle pour les divorces. Nous en présentons un à la section 8.3.4.

La première chose à faire est de réaliser une boucle sur tous les individus du modèle. On ne s'intéresse alors qu'aux individus majeurs et célibataires. Pour chaque individu, il faut alors suivre une procédure que nous pouvons découper en différentes étapes. Dans un premier temps, des paramètres sont tirés aléatoirement afin de déterminer un intervalle d'âge ainsi qu'un « intervalle de distance ». Cet intervalle de distance est en fait déterminé comme une différence maximale entre les codes INS de deux individus. En effet, les codes INS étant proches lorsque les localités le sont, nous pouvons approximer la distance entre les habitations de deux individus par la différence en valeur absolue entre leurs codes INS. Un autre entier est tiré aléatoirement : s, la taille de l'ensemble des personnes significatives de l'individu traité. Il faut ensuite définir cet ensemble des personnes significatives. Pour cela, il faut créer une deuxième boucle sur les individus et conserver les s premiers individus satisfaisant aux intervalles de distance et d'âge. Grâce à cet ensemble, il est ensuite possible de définir la pression sociale sp et l'influence de l'âge ai grâce auxquelles on détermine un nouvel intervalle d'âge. Ce nouvel intervalle est celui auquel doit satisfaire un individu afin d'être considéré

2. En effet, pour une seule année d'évolution, si nous ne divisons pas le vecteur initial, le programme ne se termine pas en 1h sur le cluster Lemaitre2. La différence de temps étant déjà tellement importante, nous n'avons pas voulu encombrer le cluster en autorisant un temps d'exécution plus important au job uniquement pour connaître le temps réel d'exécution.Avec les même paramètres (nombre de tâches, nombre de cpus par tâche, mémoire par cpu, ...) et pour la même durée d'évolution, le programme s'exécute en moins de deux minutes

comme partenaire potentiel de l'individu en cours de traitement. Parmi les personnes significatives, nous pouvons alors rechercher les partenaires potentiels. Une fois que l'on en trouve un, on vérifie si l'individu initial se trouve également dans sa tranche d'âge tolérée et si oui, on marie les deux agents.

Malheureusement, si ce modèle semble fort intéressant et assez réaliste, le temps d'exécution devient bien trop important. Nous avons donc tenté de trouver des solutions afin de le réduire.

Dans un premier temps, nous avons limité le nombre maximal de personnes significatives d'un individu à 59. Cela ne fut pas suffisant.

Nous avons alors décidé de ne plus boucler sur les individus mais sur les ménages, en ne considérant alors que les ménages dont le chef de famille est célibataire. De la sorte, nous réduisons grandement le nombre d'individus traités. Cependant, nous réduisons également le réalisme du processus. S'il n'est pas étonnant de supposer qu'un individu ait quitté le ménage de ses parents avant de se marier, VirtualBelgium n'implémente pas encore le départ des enfants du ménage. Il serait donc intéressant pour de futurs travaux de se pencher sur le problème.

Pour passer d'une boucle sur les individus à une boucle sur les ménages, quelques modifications ont dû être apportées. En effet, au sein de la boucle, les ménages sont susceptibles d'être supprimés en cas de mariage. Un simple adaptation des itérateurs de boucle résulterait en une erreur. Pour parvenir à ce problème, nous avons légèrement modifié notre processus de mariage. Désormais, les ménages ne sont pas immédiatement supprimés mais on définit leur type à *TBR* (To Be Removed). Dans nos boucles sur les ménages, il suffit alors d'ignorer les ménages marqués de la sorte. Enfin, à la fin de notre modèle, nous appelons la fonction `removeEmptyHouseholds()` qui retire tous les ménages devant l'être.

Ces modifications ont permis de réduire grandement le temps de calcul [3]. C'est donc de cette version modifiée que nous analyserons les résultats à la

3. Malheureusement, suite à un manque de temps, nous n'avons pas pu laisser tourner le premier modèle, avant modifications, sur la population complète car le job a été arrêté après 3h d'exécution. Relancer le job aurait été une perte de temps pour nous à un moment où le temps était bien trop précieux pour se le permettre.

fin de ce chapitre. Cependant, la différence en temps d'exécution par rapport au premier modèle reste extrêmement importante.

8.3.3 Troisième modèle : mariages

Pour ce troisième et dernier modèle, nous avons tenté de d'adapter les heuristiques de recherche séquentielle présentées dans le chapitre 6. Comme dans le modèle précédent, seuls les mariages sont simulés et nous devons donc également faire appel à un deuxième modèle simulant les divorces.

Le principe est que chaque année, tous les célibataires rencontrent un certain nombre de personnes leur étant suffisamment similaires. Si l'une d'entre elles leur plaît et que l'attirance est mutuelle, un nouveau couple est créé.

Plus précisément, nous bouclons sur tous les individus et nous intéressons uniquement aux célibataires de plus de 18 ans. Pour chacun d'entre eux, nous opérons une deuxième boucle sur l'ensemble des individus et ne nous arrêtons que sur les individus susceptibles de leur plaire, c'est-à-dire célibataires, majeurs et du sexe opposé. Si tel est le cas, nous vérifions si les deux individus se conviennent via un attribut `expectation` que nous avons ajouté à la classe `Individual`. Lorsque deux personnes se conviennent mutuellement, le mariage est lancé. Cependant, le nombre d'individus suffisamment similaires rencontrés est limité à 5. Chaque individu évaluera donc au maximum 5 partenaires potentiels par année.

À nouveau, ce modèle demande beaucoup trop de temps d'exécution. Nous avons donc dû le simplifier également. Pour cela, nous avons décidé de réaliser les mêmes modifications que pour le premier modèle puisqu'elles s'étaient avérées utiles. Nous avons donc transformé nos boucles sur les individus en boucles sur les ménages. Encore une fois, il faut définir un état intermédiaire *TBR* pour les ménages qui doivent être supprimés mais ne peuvent pas l'être tant que la boucle n'est pas terminée.

Le temps d'exécution de ce modèle est du même ordre que celui de modèle précédent.

8.3.4 Quatrième modèle : divorces

Nous présentons ici un modèle de divorce que nous avons imaginé afin de le combiner avec les deux modèles de mariages proposés aux sections 8.3.2 et 8.3.3. Ce modèle est assez simpliste et se base sur la mesure de dissimilarité que nous avons définie. Dans ce modèle, nous faisons l'hypothèse que des couples très dissemblables sont plus susceptibles de se séparer. Nous calculons donc la dissimilarité des deux conjoints et lui soustrayons 15. Afin d'apporter une touche aléatoire, nous ajoutons à cette soustraction un nombre tiré aléatoirement entre 0 et 2. Si le résultat est négatif, le couple divorce.

8.4 Résultats et comparaison des modèles

Dans cette section, nous comparons nos résultats obtenus pour l'année 2001 aux données réelles recueillies. Nous commençons par analyser les résultats obtenus pour les mariages avec nos trois méthodes. Ensuite, nous analysons le produit de nos deux modèles de divorces.

8.4.1 Résultats pour les mariages

Dans un premier temps, penchons-nous sur les résultats concernant les mariages. Nous voulons savoir si nos distributions de l'âge lors des mariages sont semblables à la distribution réelle. Cette distribution est déterminée via les statistiques fournies par Statbel[37].

Pour le premier modèle, nous obtenons les résultats présentés aux figures 8.1 et 8.1.

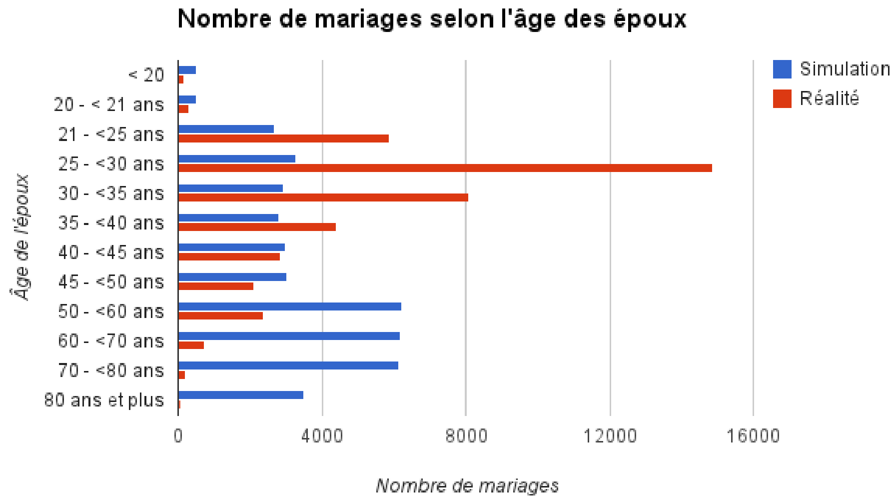

FIGURE 8.1 – Comparaison du nombre de mariages selon l'âge des époux - Modèle 1

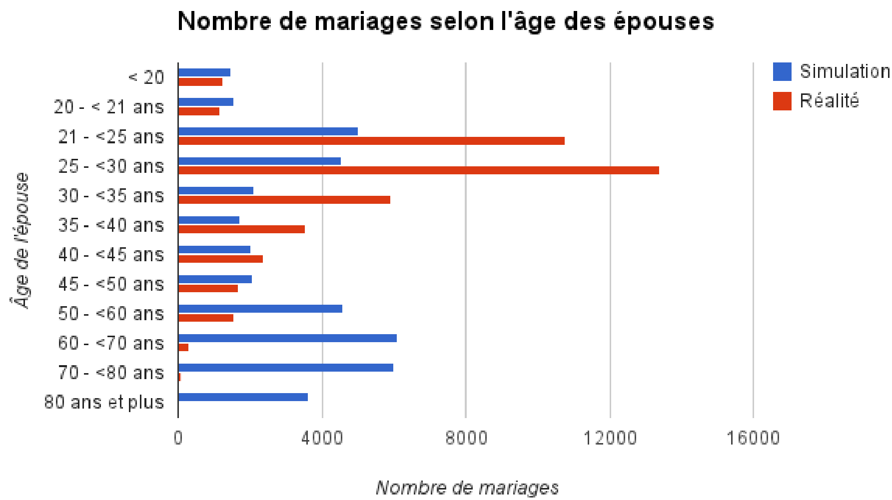

FIGURE 8.2 – Comparaison du nombre de mariages selon l'âge des épouses - Modèle 1

Nous pouvons remarquer une grande différence entre les données réelles et notre simulation. En effet, bien trop de mariages sont déclarés pour des individus d'âge mûr. Nous pouvons en effet voir sur les figures 8.1 et 8.1 que les classes d'âges supérieures à 50 ans sont sur-représentées dans notre simulation tandis que les classes d'âge moyen situé entre 21 et 40 ans sont sous-représentées et en particulier, la classe des 25-29 ans.

Étonnamment, le constant est le même pour nos trois modèles. En effet, les allures de nos simulations sont toutes similaires. Nous présentons les résultats aux figures 8.3 et 8.4pour les modèles 2 et 3 respectivement.

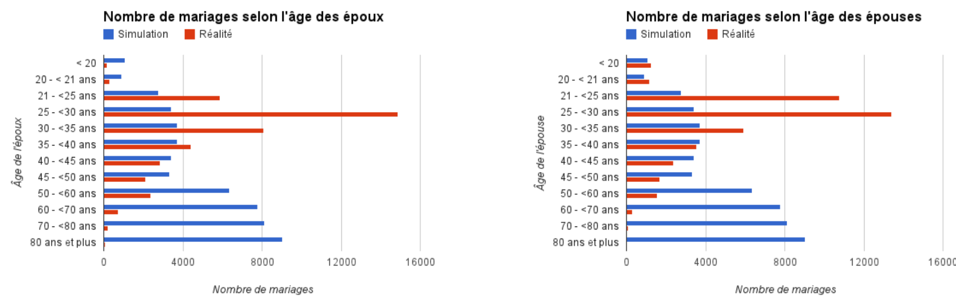

FIGURE 8.3 – Comparaison du nombre de mariages selon l'âge des époux - Modèle 2

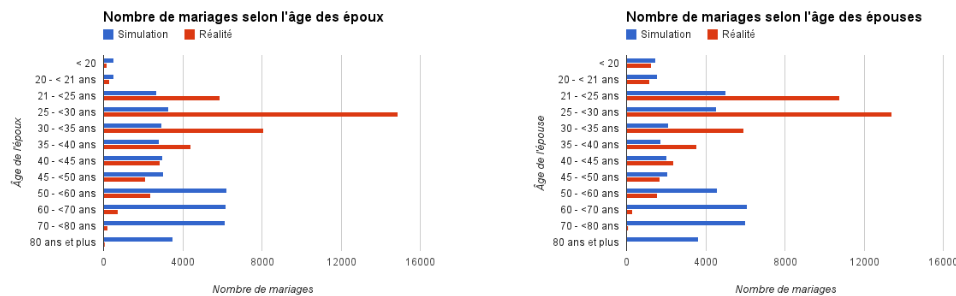

FIGURE 8.4 – Comparaison du nombre de mariages selon l'âge des époux - Modèle 3

Pour comprendre ce phénomène, nous nous sommes intéressés de plus près à la population initiale. La figure 8.5 présente la distribution des âges

de la population, restreinte aux individus déclarés comme chefs de ménages ou conjoints, selon le genre et le statut marital. Notons que nous utilisons le terme célibataire pour tous les individus n'étant pas actuellement mariés, ce terme englobe donc les veufs et les divorcés.

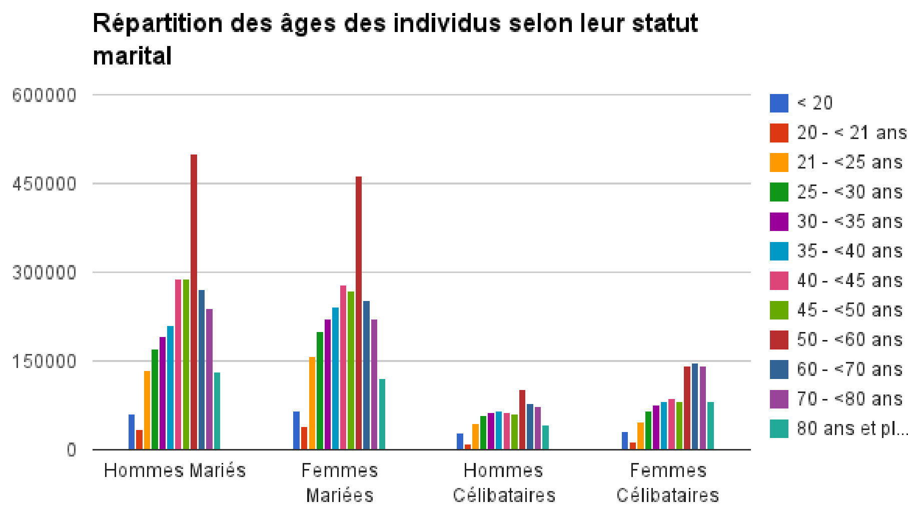

FIGURE 8.5 – Répartition des âges selon le genre et le statut marital - Chefs de ménage et conjoints uniquement

Tout d'abord, nous remarquons que les individus célibataires se trouvent plus dans les hautes tranches d'âge. Nous pouvons exprimer ce constat plus clairement à l'aide de diagrammes circulaires présentés à la figure 8.6.

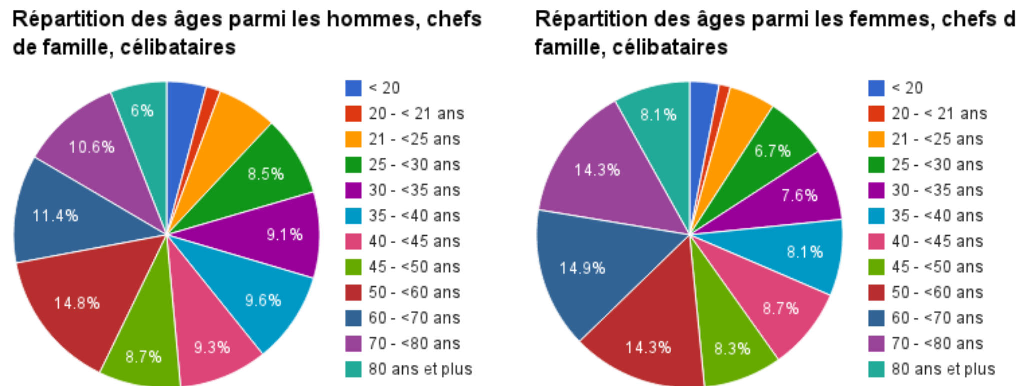

FIGURE 8.6 – Répartition des âges des chefs de ménages célibataires

Nous pouvons remarquer que la tranche des 50 ans et plus représente en fait environ la moitié des célibataires considérés par nos modèles. Nos résultats ne sont donc pas étonnants.

Une autre particularité de la figure 8.5 est que le nombre d'individus mariés est nettement supérieur au nombre de célibataires. Nous nous sommes donc interrogé sur le réalisme de cette distribution. Après quelques recherches, nous avons trouvé des statistiques fournies par l'Insee[22]. Notons que ces statistiques sont des statistiques relatives à la France et à l'année 2006 mais faute de mieux, nous nous en conterons. Ces statistiques indiquent que 51.1% des hommes et 46.5% des femmes de 15 ans ou plus sont mariés. Dans notre population, nous avons pourtant 78% et 70% d'hommes et femmes mariés.

Rappelons que nous n'avons ici considérés que les célibataires étant chefs de famille, cette simplification du modèle est donc peut-être la cause de ce problème. Pour nous en assurer, nous pouvons réaliser le même graphique en considérant cette fois tous les individus de 15 ans ou plus. Nous obtenons alors les résultats présentés à la figure 8.7.

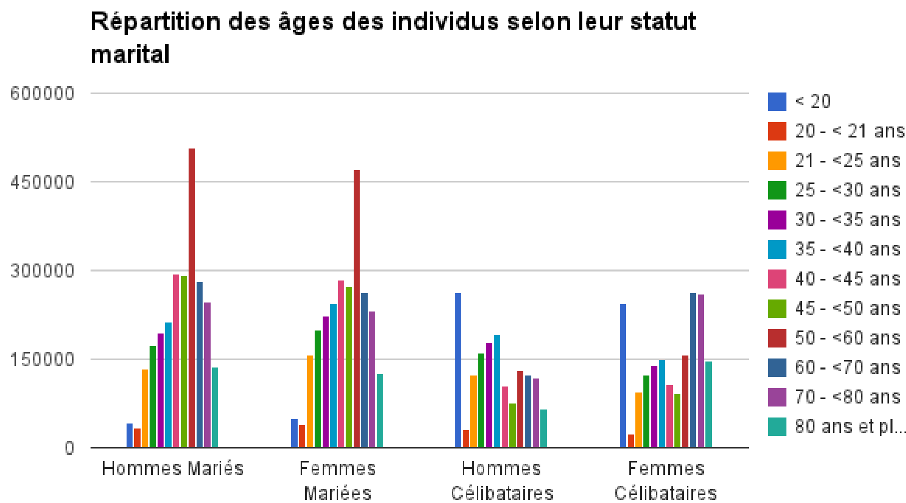

FIGURE 8.7 – Répartition des âges selon le genre et le statut marital - Individus de 15 ans et plus

Bien que la distribution soit plus équilibrée, nous constatons encore une majorité d'individus mariés qui représentent environ 60% des deux sexes.

Nous pouvons à nouveau examiner les distributions des âges parmi les célibataires uniquement que nous présentons à la figure 8.8.

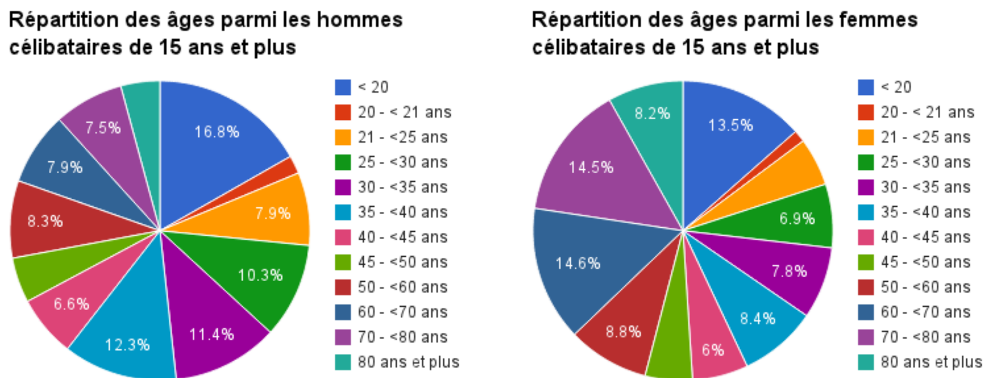

FIGURE 8.8 – Répartition des individus célibataires de 15 ans et plus

La dynamique est différente de la précédente. Nous pouvons observer que la tranche des 50 ans et plus représente un peu moins de 30% chez les hommes. Cependant, elle représente toujours près de la moitié des femmes.

Notre simplification distord donc la population synthétique initiale mais dans une mesure assez raisonnable. La population initiale complète est fort distincte de la réalité, ce qui explique le manque de réalisme de nos résultats.

Après avoir analysé les distributions d'âge, nous nous sommes intéressés aux couples formés via l'âge de chaque conjoint. Nous avons alors représenté les données comme une surface. La figure 8.9 représente la répartition des mariages selon les âges des conjoints obtenue avec le premier modèle.

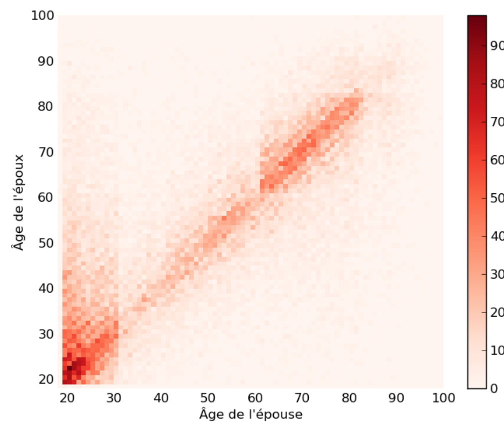

FIGURE 8.9 – Répartition des mariages selon l'âge des conjoints – Modèle 1

Cette figure se lit de la manière suivante : plus le point est sombre, plus il y a eu de mariages entre deux individus des âges correspondants. Nous voyons très distinctement une diagonale se former. Ce résultat semble assez convainquant et intuitif puisqu'il signifie que les individus sont généralement mariés à quelqu'un de leur tranche d'âge.

En réalisant le même graphique pour le deuxième modèle, nous obtenons à la figure 8.10 une diagonale bien plus prononcée. En réalité tous les mariages ont été réalisés entre personnes du même âge exactement. Cela signifierait que notre définition de l'intervalle d'âge acceptable pour les partenaires potentiels

est trop stricte. Cela peut également expliquer les résultats de distribution des âges qui étaient encore plus marqués que les autres modèles dans les hautes tranches d'âge.

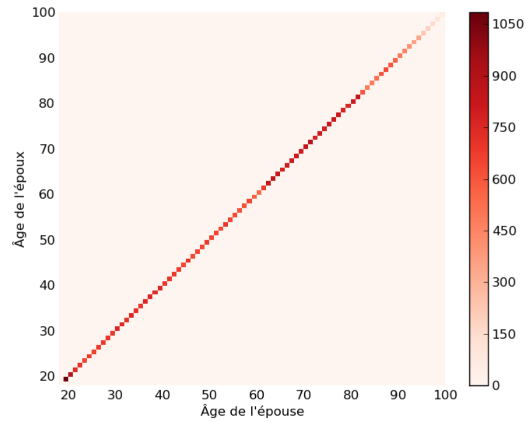

FIGURE 8.10 – Répartition des mariages selon l'âge des conjoints – Modèle 2

Notre troisième modèle donne des résultats du même type que le premier mais crée des écarts un peu plus importants et on remarque également une plus grande symétrie. Ces résultats sont présentés à la figure 8.11.

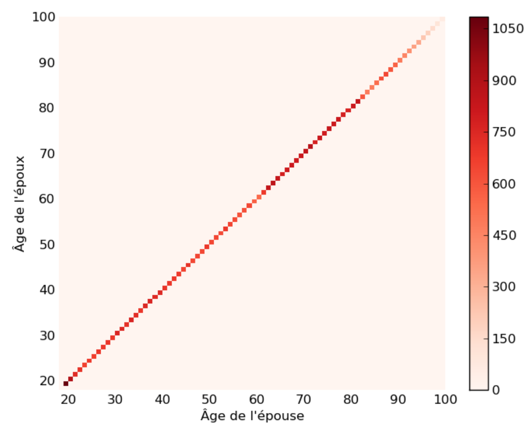

FIGURE 8.11 – Répartition des mariages selon l'âge des conjoints – Modèle 3

8.5 Résultats pour les divorces

Lorsque nous avons appliqué notre premier modèle, nous avons réalisé qu'il génère un nombre démesuré de divorces (environ 200 000). Nous avons donc décidé d'ajouter un paramètre très important afin de réduire le nombre de divorces. Cependant, cette solution n'est pas très réfléchie et nous ne la trouvons pas très élégante. Malheureusement aucune autre méthode ne nous est venue à l'esprit pour résoudre le problème. Nous avons donc décidé de laisser cette méthode de côté.

Le nombre exact de divorces ayant été déclarés en Belgique en 2001 est de 29 314. Le deuxième modèle des divorces en calcule 42 358 lors de la première année d'exécution mais se stabilise ensuite avec 32 641 divorces dans la deuxième année et 31 711 pour la troisième. Le nombre plus important de divorces lors de la première année peut encore une fois s'expliquer par la structure de la population synthétique initiale. En effet, nous pouvons y retrouver des ménages très surprenants avec des différences d'âge explosant tous les records ou même la présence d'enfants déclarés mariés. Ainsi, on retrouve parmi les divorces générés lors de la première année d'exécution un couple composé d'individus âgés de 2 et 77 ans.

Nous avons donc voulu représenté la répartition des âges dans les couples comme dans notre section précédente. Les résultats obtenus sont présentés à la figure 8.12. On peut y remarquer que les cas extrêmes tels que celui que nous venons de mentionner sont en réalité assez rares. Nous pouvons également voir que la zone la plus représentée du graphe a une forme particulière qui suggère que la population a été générée selon des classes d'âges plutôt que directement par des âges précis. Notons que pour cette dernière figure, nous n'avons pas restreint les intervalles d'âge à plus de 18 comme pour les figures précédentes car plusieurs mineurs sont déclarés mariés.

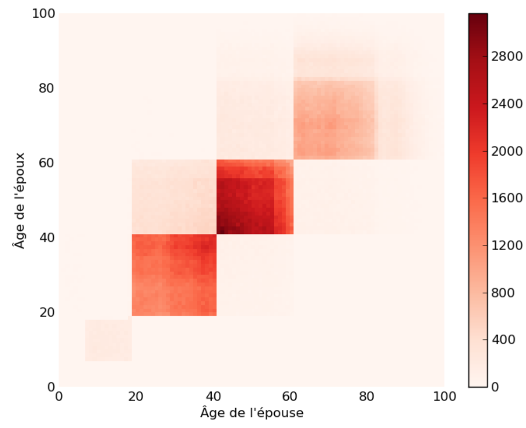

FIGURE 8.12 – Répartition des mariages selon l'âge des conjoints – Modèle 3

Conclusion

Au cours de notre étude du problème des mariages et divorces, nous avons pu découvrir différents types de modèles. Nous avons examiné un modèle de choix discret, un modèle de recherche séquentielle ainsi qu'un modèle définissant des sous-ensembles d'individus à traiter.

Lors de ces analyses, nous avons pu noter la difficulté que représentent la modélisation de tels phénomènes, la nécessité d'allier la simplicité et le réalisme.

Ces différents modèles ont été implémentés avec succès dans VirtualBelgium. Certaines modifications ont toutefois dû être apportées afin de limiter les ressources nécessaires au programme. Si les résultats obtenus ne sont pas toujours ceux attendus, le travail restant est grandement diminué puisqu'il faut désormais les calibrer mais les codes existent déjà. Ces codes tournent sans erreur sur des architectures de calcul de haute performance mais peuvent également être lancés sur un simple ordinateur personnel à condition d'adapter la taille de la population.

Lors de l'analyse des résultats, nous avons pu remarqué des points positifs comme des négatifs. Nous avons également mis en évidence l'importance qu'a la population initiale dont la moindre distorsion par rapport à la réalité peut avoir de grandes conséquences. Nous avons également remarquer que malgré que les trois modèles aient des approches assez différentes, les résultats obtenus sont qualitativement similaires, du moins pour les paramètres analysés.

Notons qu'au cours de nos analyses, nous avons pu observer quelques incohérence dans la population initiale de VirtualBelgium. Nous avons pu remarquer que des individus mineurs peuvent être déclarés en couple. De

même, certains individus mineurs sont déclarés comme chefs de leur ménage. Si cette situation est vraisemblable dans le cadre d'enfants émancipés, il faut quand même introduire une limite inférieure des âges et éviter d'avoir des individus de 6 ans chefs de leur ménage. Nous avons également remarqué qu'un pourcentage trop important de la population est déclaré marié.

Annexe A

Principes de programmation orientée objet et bases de C++

Le projet VirtualBelgium étant implémenté en C++, il nous a paru intéressant de rappeler quelques notions de programmation orientée objets et, en particulier, certains mots de vocabulaire, pour permettre une complète compréhension de ce mémoire. Cette annexe s'est essentiellement basée sur le manuel de J.-F. Rabasse [33] et le support du cours de *Conception et Programmation orientées objets* de P.Heymans [20].

A.1 Classes et instances

Le principe d'objet en programmation est assez similaire à la définition classique d'un objet. En programmation orientée objets, un objet est une entité informatique possédant des *attributs*, les données de cet objet, et des *méthodes*, les fonctions permettant de le manipuler. Les méthodes regroupent les opérations qui concernent l'objet, on parle des opérations CRUD : Create, Read, Udate, Delete. Chaque objet est appelé *instance* d'une classe, c'est-à-dire qu'il est une « réalisation » d'un type d'objet. Ces définitions sont très vagues et méritent d'être clarifiées. Pour cela, nous allons prendre un exemple.

Une *classe* est un type d'objet, imaginons donc une classe `Maison`. Toutes les maisons ont certains paramètres les décrivant, nous pourrions par exemple avoir comme *attributs* la surface habitable, le nombre de chambres et l'existence ou non d'un jardin.

Ma maison n'étant pas celle de mon voisin, toutes deux sont des *instances* de la classe `Maison` (et cela même si elles avaient exactement les mêmes paramètres). Enfin, je peux construire une extension ou demander des informations sur une maison que je désire acheter, c'est là qu'interviennent les *méthodes* de la classe `Maison` qui permettent de recueillir ou modifier les attributs d'une instance de la classe.

On appelle une *interface de classe* la description d'une classe destinée au compilateur. On la retrouve dans un fichier *header* portant le nom de la classe.
Pour notre classe `Maison`, nous pourrions avoir l'interface suivante [1].

```
1  class Maison
2  {
3    public :
4      float getSurfaceHabitable();
5      void construireExtension(float tailleExtension);
6
7    private:
8      float surfaceHabitable;
9      float nombreChambres;
10     int jardin;
11 }
```

Nous avons ici des méthodes correspondant aux Read et Update du CRUD mais il ne faut pas oublier les Create et Delete. Il faut alors définir un *constructeur* (ou plus [2]) et un *destructeur* pour la classe.

```
1  class Maison
2  {
3    public :
4      Maison(); //constructeur par defaut
5      Maison(float surfaceHab); //constructeur specifiant la
          surface habitable de l'objet
6      ~Maison(); //destructeur
7      float getSurfaceHabitable();
```

1. Dans cet exemple, les méthodes sont peu nombreuses pour simplifier la lecture.
2. Plusieurs constructeurs spécifiant différents attributs peuvent être proposés.

```
8       void construireExtension(float tailleExtension);
9
10    private:
11      float surfaceHabitable;
12      float nombreChambre;
13      int jardin;
14  }
```

En plus de l'interface de la classe, il faut évidemment implémenter ses différentes méthodes. Dans notre exemple, nous aurions par exemple un fichier **Maison.cpp** contenant

```
1  Maison::Maison()
2  {
3    surfaceHabitable = 0;
4    jardin = 0;
5    nombreChambres = 0;
6  }
7
8  Maison::Maison(float surfaceHab)
9  {
10   surfaceHabitable = surfaceHab;
11   jardin = 0;
12   nombreChambres = 0;
13 }
14
15 Maison::~Maison()
16 {
17 }
18
19 float Maison::getSurfaceHabitable()
20 {
21   return surfaceHabitable;
22 }
23
24 void Maison::construireExtension(float tailleExtension)
25 {
26   surfaceHabitable += tailleExtension;
27 }
```

Créer une instance d'une classe se fait très simplement, comme on créerait une variable d'un certain type en programmation classique. Ensuite, contrairement aux fonctions classiques qui sont simplement appelées, les méthodes sont appliquées *sur* l'instance qui le demandent.

```
int main()
{
  Maison maMaison, celleDuVoisin;

  float s1,s2;

  s1 = maMaison.getSurfaceHabitable();
  s2 = celleDuVoisin.getSurfaceHabitable();

  return 0;
}
```

On remarque bien dans cet exemple que le nom de la méthode ne suffit pas, il faut préciser à quelle instance on souhaite l'appliquer. En effet, il n'y a aucune raison pour que la maison de mon voisin ait la même surface habitable que la mienne.

Un mécanisme important de la programmation orientée objets est *l'encapsulation de données*. Ce mécanisme a été introduit sans être mentionné dans l'exemple de l'interface de la classe `Maison` avec les *attributs de visibilité* **public** et **private**. Ces attributs sont destinés au compilateur afin de prévenir les appels non autorisés à des méthodes ou des attributs.
En effet, si l'implémentation des méthodes d'une classe a accès à toutes ses méthodes et attributs (publiques ou privés), ce n'est pas le cas du code externe qui n'a accès qu'aux méthodes publiques. Ainsi, si dans notre dernier exemple nous avions voulu, dans la fonction `main`, trouver la surface habitable de ma maison par l'attribut `surfaceHabitable` sans passer par la méthode publique `getSurfaceHabitable`, nous aurions eu une erreur de compilation.

Au premier abord, ce mécanisme peut paraître restrictif mais il est, en réalité, un mécanisme de sécurité pour les raisons suivantes.

Tout d'abord, l'encapsulation de données permet d'utiliser une classe en

ne s'intéressant qu'à son interface sans regarder son implémentation. Cela permet donc de modifier l'implémentation d'une classe (mais en conservant l'interface plublique) sans devoir modifier tous les appels externes à cette classe.

De plus, certains attributs d'un objet peuvent être liés entre eux, la modification de l'un exigerait alors une mise à jour des autres. En interdisant au code utilisateur d'une classe d'avoir un accès direct aux attributs d'un objet, nous en assurons la cohérence.

A.2 Héritage

Si maintenant je voulais ajouter une classe `Appartement` à mon programme, je me rendrais vite compte que certains attributs comme `nombreChambres` et `surfaceHabitable` sont également nécessaires, tandis qu'un attribut tel que l'étage du bien ne s'applique qu'à `Appartement` et pas à `Maison`. Dans ce cas, il serait dommage de recopier le code associé aux attributs communs.

On peut alors créer une nouvelle classe `Habitation` définie par les attributs communs de `Maison` et `Appartement`.
Ensuite, on construit les classes dérivées qui *héritent* de toutes les méthodes d'`Habitation` et ajoute leurs spécificités.

Nous aurions alors les interfaces suivantes.

```
1  class Habitation
2  {
3    public :
4      Habitation();
5      Habitation(float surfaceHab);
6      float getSurfaceHabitable();
7
8    private:
9      float surfaceHabitable;
10     float nombreChambres;
11 }
```

```
1  class Appartement : public Habitation //declaration d'
       heritage
2  {
3    public :
4      Appartement();
5      Appartement(float surfaceHab);
6      float getSurfaceHabitable();
7      int getEtage();
8
9    private:
10     int etage;
11 }
```

Désormais, dans la clase `Appartement`, seul l'attribut `etage` apparaît.
On dira que la classe `Habitation` est **la** *superclasse* d'`Appartement` et, inversément, `Appartement` est **une** sousclasse d'`Habitation`.

A.3 Polymorphisme et classes abstraites

Une classe abstraite est une classe qui possède au moins une méthode virtuelle pure, c'est-à-dire une méthode qui n'est pas implémentée mais devra l'être par les sous-classes.
Les classes abstraites ont la particularité qu'elles ne peuvent être instanciées. En effet, s'il on pouvait créer une instance de la classe, on pourrait alors essayer de lui appliquer la méthode virtuelle pure, ce qui n'est pas acceptable puisque la méthode n'a pas été implémentée.
Dans notre exemple, nous pourrions ajouter une méthode virtuelle pure `description`[3] à notre classe `Habitation` et obtenir l'interface suivante

```
1  class Habitation
2  {
3    public :
4      Habitation();
5      Habitation(float surfaceHab);
6      float getSurfaceHabitable();
```

3. Décrire une habitation sans en donner le type (appartement, maison, ...) n'aurait que peu de sens.

```
7      virtual void description() = 0; //syntaxe de
          declaration d'une methode virtuelle pure
8
9   private:
10     float surfaceHabitable;
11     float nombreChambres;
12 }
```

Transformer la classe `Habitation` en classe abstraite (et donc ininstanciable) peut se justifier par le fait qu'une habitation est juste un concept mais n'est pas un objet, ce qui est souvent le cas des classe abstraites.

Annexe B

Codes pour les simulations : recherche unilatérale

B.1 Code principal

```
1  # coding: utf8
2
3  import random
4  from pprint import pprint
5  import matplotlib.pyplot as plt
6  import numpy as np
7  from functions import *
8  from f_oneSided import *
9
10 N = 1000 #population size
11 Length = int(0.95 * N)
12 NSimulations = 10000
13
14 best_mate = [0] * Length
15 top_10 = [0] * Length
16 top_25 = [0] * Length
17 worst_25 = [0] * Length
18
19 distrib = 'uniform'  #ditrib must be set to 'uniform' or '
       norm'
20 #param of the normal distribution if distrib set to 'norm'
```

```
21 mu = 50
22 sigma = 10
23
24 for simulation in range(NSimulations):
25     print "Simulation ",simulation,"/",NSimulations
26
27     for length in range(Length):
28
29     #createPop(distrib,N,women)
30     if distrib == 'uniform':
31         women = range(N
32     else:
33         women = []
34       for i in range(N):
35           women.append(random.normalvariate(mu, sigma))
36
37       expectation = 0
38       random.shuffle(women)
39
40       #adolescence
41       for i in range(length):
42       # update man's expectation
43       expectation = max(expectation,women[i])
44
45       found = 0
46     i = length
47       while (not found) and (i < N):
48         #if woman is good enough stop
49         if women[i]>=expectation:
50              obtained = women[i]
51              found = 1
52       i += 1
53     if found == 0:
54           obtained = women[-1]
55
56       if distrib == 'uniform':
57         updateParamUniform(obtained,worst_25,top_25,top_10,
              best_mate,N,length)
58       else:
```

```
59            updateParamNorm(obtained,worst_25,top_25,top_10,
                best_mate,women,length)
60
61  best_mate = [100*x/float(NSimulations) for x in best_mate]
62  top_10 = [100*x/float(NSimulations) for x in top_10]
63  top_25 = [100*x/float(NSimulations) for x in top_25]
64  worst_25 = [100*x/float(NSimulations) for x in worst_25]
65
66  plt.figure()
67  plt.plot(range(Length),best_mate,'^',label='best mate')
68  plt.plot(range(Length),top_25,'-',label='top 10')
69  plt.plot(range(Length),top_10,'--',label='top 25')
70  plt.plot(range(Length),worst_25,':',label='worst 25')
71  plt.xlabel(u"Pourcentage de candidats examines dans la
        premiere phase")
72  plt.ylabel(u'Pourcentage de partenaires choisis de chaque
        categorie')
73  plt.legend()
74  plt.savefig('oneSided.png')
```

B.2 Fonctions utilisées

```
1  import random
2  from pprint import pprint
3  import matplotlib.pyplot as plt
4  import numpy as np
5  from functions import *
6
7  def updateParamUniform(obtained,worst_25,top_25,top_10,
       best_mate,N,length):
8       if obtained < 0.25*N:
9             worst_25[length] += 1
10      elif obtained >= 0.75*N:
11            top_25[length] += 1
12          if obtained >= 0.90*N:
13          top_10[length] += 1
14        if obtained == N-1:
15              best_mate[length] += 1
16
17
```

```
18 def updateParamNorm(obtained,worst_25,top_25,top_10,
       best_mate,women,length):
19         if obtained < np.percentile(women,25):
20                 worst_25[length] += 1
21         elif obtained >= np.percentile(women,75):
22                 top_25[length] += 1
23                 if obtained >= np.percentile(women,90):
24                         top_10[length] += 1
25                         if obtained == max(women):
26                                 best_mate[length] += 1
```

Annexe C

Codes pour les simulations : recherche bilatérale

C.1 Code principal

```
import matplotlib.pyplot as plt
from functions import *

N = 100 # number of individual of each gender
NSimulations = 100
Max_Times = 80
teenage_times = range(Max_Times)
distrib = 'uniform'
#distrib must be set to 'uniform' or 'norm'
methods = ['TNB','Humble','Adjust1','Adjust2','Adjust3']

#param of the normal distribution if distrib set to 'norm'
mu = 50
sigma = 10

marriedProportion = {'TNB': [0]* Max_Times,
                     'Humble': [0]* Max_Times,
                     'Adjust1': [0]* Max_Times,
                     'Adjust2': [0]* Max_Times,
                     'Adjust3': [0]* Max_Times
                    }
```

```
22 mean_value = {'TNB': [0]* Max_Times,
23               'Humble': [0]* Max_Times,
24               'Adjust1': [0]* Max_Times,
25               'Adjust2': [0]* Max_Times,
26               'Adjust3': [0]* Max_Times
27              }
28 mean_difference = {'TNB': [0]* Max_Times,
29                    'Humble': [0]* Max_Times,
30                    'Adjust1': [0]* Max_Times,
31                    'Adjust2': [0]* Max_Times,
32                    'Adjust3': [0]* Max_Times
33                   }
34
35 divideBy = {'TNB': [0]* Max_Times,
36             'Humble': [0]* Max_Times,
37             'Adjust1': [0]* Max_Times,
38             'Adjust2': [0]* Max_Times,
39             'Adjust3': [0]* Max_Times
40             }
41
42 for simulation in range(NSimulations):
43     print "Simulation ",simulation+1,"/",NSimulations
44     for method in methods:
45         print "Method :",method
46
47         #Initialize lists to [], the lists will contain a
               value for each number of individuals that were
               checked in order to set the expectation value
48         marriedProportions = []
49         mean_values = []
50         mean_differences = []
51
52         for teenageLength in teenage_times:
53             value_differences = []
54             mate_values = []
55
56             #create initial population
57             men = []
58             women = []
59             create_pop(N,men,women, distrib ,mu, sigma)
```

```
60
61              #teenage time..
62              adolescence(teenageLength,men,women,method,N)
63
64              #then comes the meeting time...
65              ok = 1
66              while ok:
67              #ok if still a man that hasn't proposed to a
                    free woman yet
68                  //shuffle randomly
69                  random.shuffle(women)
70
71                  ok = 0
72                  i = 0
73                  while i < len(men):
74                      #test if men[i] hasn't already proposed
                            to women[i]
75                      if women[i]['id'] not in men[i]['
                            alreadyProposedTo']:
76                          #test if men[i] and women[i] want
                                to be together
77                          #if so, remove them from the single
                                people pool
78                          if coupleOk(men[i],women[i]):
79                              #keep in memory their mate
                                    values and the difference
                                    between them
80                              mate_values.append(men[i]['
                                    value'])
81                              mate_values.append(women[i]['
                                    value'])
82                              value_differences.append(abs(
                                    men[i]['value']-women[i]['
                                    value']))
83                              men.remove(men[i])
84                              women.remove(women[i])
85                          else:
86                              #else, add women[i] to the list
                                    of women men[i] already
                                    proposed to
```

```
87                              men[i]['alreadyProposedTo'].
                                    append(women[i]['id'])
88
89                              # if men[i] has not been
                                    removed, check if he still
                                    has a partner he has not
                                    proposed to
90
91                              #recompute ok
92                              j=0
93                              while ((not ok) and (j < len(
                                    women))):
94                                  ok = women[j]['id'] not in
                                        men[i]['
                                        alreadyProposedTo']
95                                  j += 1
96
97                      i += 1
98
99              marriedProportion[method][teenageLength] += N-
                    len(men)
100             if len(mate_values) > 0:
101                 mean_value[method][teenageLength] += np.
                        mean(mate_values)
102                 mean_difference[method][teenageLength] +=
                        np.mean(value_differences)
103                 divideBy[method][teenageLength] += 1
104
105 #Only to avoid dividing by 0
106 for method in methods:
107     for i in range(len(divideBy[method])):
108         if divideBy[method][i] == 0:
109             divideBy[method][i] = 1
110
111 ###End of all simulations: divide by the number of
        simulations
112 for method in methods:
113     marriedProportion[method] = [ x/float(NSimulations) for
            x in marriedProportion[method]]
```

```
114     mean_value[method] = [ x/float(y) for x,y in zip(
            mean_value[method],divideBy[method])]
115     mean_difference[method] = [ x/float(y) for x,y in zip(
            mean_difference[method],divideBy[method])]
116
117 ###plots
118 plt.figure()
119 for method in methods:
120     plt.plot(teenage_times,marriedProportion[method],label=
            method)
121     plt.grid()
122     plt.title('Married proportion of the population')
123     plt.legend()
124 plt.savefig('MarriedProp.png')
125
126 plt.figure()
127 for method in methods:
128     plt.plot(teenage_times,mean_difference[method],label=
            method)
129     plt.grid()
130     plt.title('Mean difference between mates')
131     plt.legend()
132 plt.savefig('Mean-difference.png')
133
134 plt.figure()
135 for method in methods:
136     plt.plot(teenage_times,mean_value[method],label=method)
137     plt.grid()
138     plt.title('Mean Value of the mates')
139     plt.legend()
140 plt.savefig('Mean-value.png')
```

C.2 Fonctions utilisées

```
1 import random
2 import numpy as np
3 import math
4 from mwmatching import *
5 ###mwmatching implements a maximum cardinality matching
      algorithm -- downloaded from http://jorisvr.nl/
```

```
    maximummatching.html

def create_pop(N,men,women,distrib,mu,sigma):
    if distrib == 'uniform':
        create_pop_uniform(N,men,women)
    else:
        create_pop_norm(N,men,women,mu,sigma)

def create_pop_uniform(N,men,women):
    for i in range(N):
        man = { 'id': i,
                'value': i+1, #this way the distribution of
                    mate_values will be U[0,100]
                'expectation' : 0,
                'alreadyProposedTo': []
              }
        woman = { 'id': N+i,
                  'value': i+1, #this way the distribution
                      of mate_values will be U[0,100]
                  'expectation' : 0,
                  'alreadyProposedTo': []
                }
        men.append(man)
        women.append(woman)

def create_pop_norm(N,men,women,mu,sigma):
    for i in range(N):
        man = { 'id': i,
                'value': random.normalvariate(mu, sigma),
                'expectation' : 0,
                'alreadyProposedTo': []
              }
        men.append(man)
    for i in range(N):
        woman = { 'id': N+i,
                  'value': random.normalvariate(mu, sigma),
                  'expectation' : 0,
                  'alreadyProposedTo': []
                }
        women.append(woman)
```

```
43
44 def propose(a,b):
45     #returns 1 if b meets a's expectations, 0 else
46     return b['value'] >= a['expectation']
47
48 def coupleOk(a,b):
49     #returns 1 if both a and b are satisfied with the other
           one's value
50     return propose(a,b) and propose(b,a)
51
52 def adolescence(length, men, women, method, N):
53     if (method == 'TNB'):
54          adolescence_TNB(length, men, women, N)
55     elif (method == 'Humble'):
56         adolescence_humble(length,men,women, N)
57     elif (method == 'Adjust1'):
58         adolescence_adjust1(length, men, women, N)
59     elif (method == 'Adjust2'):
60         adolescence_adjust2(length, men, women, N)
61     elif (method == 'Adjust3'):
62         adolescence_adjust3(length, men, women, N)
63     else:
64         print 'methode non existente'
65
66 def sortdict(d):
67     s = d.items()
68     s.sort()
69     return s
70
71 def adolescence_TNB(length, men, women, N):
72     edges = []
73     for i in range(len(men)):
74         for j in range(len(women)):
75             edges.append((men[i]['id'],women[j]['id'],1))
76
77     for year in range(length):
78         random.shuffle(edges)
79         matching = maxWeightMatching(edges, True)
80         for i in range(len(matching)):
81             if i<N:
```

```
82                  if (matching[i] != -1):
83                      j = matching[i]-N
84                      men[i]['expectation'] = max(men[i]['
                            expectation'],women[j]['value'])
85                      women[j]['expectation'] = max(women[j][
                            'expectation'],men[i]['value'])
86                      men[i]['alreadyProposedTo'].append(
                            women[j]['id'])
87                      edges.remove((men[i]['id'],women[j]['id
                            '],1))
88                  else:
89                      print "No maximal matching found"
90
91
92  def adolescence_humble(length,men,women, N):
93      for i in range(len(men)):
94          men[i]['expectation']=men[i]['value']-5
95          women[i]['expectation']=women[i]['value']-5
96
97      edges = []
98      for i in range(len(men)):
99          for j in range(len(women)):
100             edges.append((men[i]['id'],women[j]['id'],1))
101
102     for year in range(length):
103           random.shuffle(edges)
104           matching = maxWeightMatching(edges, True)
105         for i in range(len(matching)):
106             if i<N:
107                 if (matching[i] != -1):
108                     j = matching[i]-N
109                     men[i]['alreadyProposedTo'].append(
                            women[j]['id'])
110                     edges.remove((men[i]['id'],women[j]['id
                            '],1))
111                 else:
112                     print "No maximal matching found"
113
114
115 def adolescence_adjust1(length, men, women, N):
```

```
116     for i in range(len(men)):
117         men[i]['expectation'] = 50
118         women[i]['expectation'] = 50
119 
120     edges = []
121     for i in range(len(men)):
122         for j in range(len(women)):
123             edges.append((men[i]['id'],women[j]['id'],1))
124 
125     for year in range(length):
126           random.shuffle(edges)
127           matching = maxWeightMatching(edges, True)
128         for i in range(len(matching)):
129             if i<N:
130                 if (matching[i] != -1):
131                     j = matching[i]-N
132                     men[i]['alreadyProposedTo'].append(
                            women[j]['id'])
133                     adjust_self_value_1(men,women,i,j,
                            length)
134                     edges.remove((men[i]['id'],women[j]['id
                            '],1))
135                 else:
136                     print "No maximal matching found"
137 
138 def adjust_self_value_1(men,women,i,j,length):
139     adjustment = 50.0/(1+length)
140     #if women[j] meets men[i]'s expectations, he proposes
141     #women[j]'s self-perceived value grows
142     if (men[i]['expectation'] <= women[j]['value']):
143         women[j]['expectation'] += adjustment
144     else:
145         women[j]['expectation'] -= adjustment
146 
147     #same thing but with inversed roles
148     if (women[j]['expectation'] <= men[i]['value']):
149         men[i]['expectation'] += adjustment
150     else:
151         men[i]['expectation'] -= adjustment
152 
```

```
153 def adolescence_adjust2(length, men, women, N):
154     for i in range(len(men)):
155         men[i]['expectation'] = 50
156         women[i]['expectation'] = 50
157 
158     edges = []
159     for i in range(len(men)):
160         for j in range(len(women)):
161             edges.append((men[i]['id'],women[j]['id'],1))
162 
163     for year in range(length):
164           random.shuffle(edges)
165         matching = maxWeightMatching(edges, True)
166         for i in range(len(matching)):
167             if i<N:
168                 if (matching[i] != -1):
169                     j = matching[i]-N
170                     men[i]['alreadyProposedTo'].append(
                            women[j]['id'])
171                     adjust_self_value_2(men,women,i,j,
                            length)
172                     edges.remove((men[i]['id'],women[j]['id
                            '],1))
173                 else:
174                     print "No maximal matching found"
175 
176 def adjust_self_value_2(men,women,i,j,length):
177     adjustment = 50.0/(1+length)
178     #if women[j] meets men[i]'s expectations, he proposes
179     if (men[i]['expectation'] <= women[j]['value']):
180         #if men[i] has a higher value than women[j] thinks
                she has,
181         #then she has to adjust her self-perceived value
182         if (men[i]['value'] > women[j]['expectation']):
183             women[j]['expectation'] += adjustment
184     else:
185         #if men[i] does not propose and his value is lower
186         #than the women[j]'s self-perceived value
187         #then she has to adjust it
188         if (men[i]['value'] <= women[j]['expectation']):
```

```
189                 women[j]['expectation'] -= adjustment
190 
191         #same thing but with inversed roles
192         if (women[j]['expectation'] <= men[i]['value']):
193             #if men[i] has a higher value than women[j] thinks
                    she has,
194             #then she has to adjust her self-perceived value
195             if (women[j]['value'] > men[i]['expectation']):
196                 men[i]['expectation'] += adjustment
197         else:
198             #if men[i] does not propose and his value is lower
199             #than the women[j]'s self-perceived value
200             #then she has to adjust it
201             if (women[j]['value'] <= men[i]['expectation']):
202                 men[i]['expectation'] -= adjustment
203 
204 def adolescence_adjust3(length, men, women,N):
205     for i in range(len(men)):
206         men[i]['expectation'] = 50
207         women[i]['expectation'] = 50
208 
209     edges = []
210     for i in range(len(men)):
211         for j in range(len(women)):
212             edges.append((men[i]['id'],women[j]['id'],1))
213 
214     for year in range(length):
215         random.shuffle(edges)
216         matching = maxWeightMatching(edges, True)
217         for i in range(len(matching)):
218             if i<N:
219                 if (matching[i] != -1):
220                     j = matching[i]-N
221                     men[i]['alreadyProposedTo'].append(
                            women[j]['id'])
222                     adjust_self_value_3(men,women,i,j)
223                     edges.remove((men[i]['id'],women[j]['id
                            '],1))
224                 else:
225                     print "No maximal matching found"
```

```
226
227 def adjust_self_value_3(men,women,i,j):
228     #if women[j] meets men[i]'s expectations, he proposes
229     if (men[i]['expectation'] <= women[j]['value']):
230         #if men[i] has a higher value than women[j] thinks
                she has,
231         #then she has to adjust her self-perceived value
232         if (men[i]['value'] > women[j]['expectation']):
233             adjustment = (men[i]['value'] - women[j]['
                    expectation'])/2.0
234             women[j]['expectation'] += adjustment
235     else:
236         #if men[i] does not propose and his value is lower
237         #than the women[j]'s self-perceived value
238         #then she has to adjust it
239         if (men[i]['value'] < women[j]['expectation']):
240             adjustment = (women[j]['expectation'] - men[i][
                    'value'])/2.0
241             women[j]['expectation'] -= adjustment
242
243     #same thing but with inversed roles
244     if (women[j]['expectation'] <= men[i]['value']):
245         #if men[i] has a higher value than women[j] thinks
                she has,
246         #then she has to adjust her self-perceived value
247         if (women[j]['value'] > men[i]['expectation']):
248             adjustment = (women[j]['value'] - men[i]['
                    expectation'])/2.0
249             men[i]['expectation'] += adjustment
250     else:
251         #if men[i] does not propose and his value is lower
252         #than the women[j]'s self-perceived value
253         #then she has to adjust it
254         if (women[j]['value'] <= men[i]['expectation']):
255             adjustment = (men[i]['expectation'] - women[j][
                    'value'])/2.0
256             men[i]['expectation'] -= adjustment
```

Annexe D

Installation de VirtualBelgium

D.1 Installation

La section suivante, écrite en collaboration avec J. Barthelemy, décrit le processus d'installation du programme VirtualBelgium sur un ordinateur personnel. Plus précisément, ce document explique comment utiliser et installer VirtualBelgium sur une architecture GNU/linux 64 bits. Pour commencer, nous détaillerons les fichiers et librairies à télécharger. Ensuite, nous décrirons les commandes d'installation à entrer et enfin celles pour lancer le programme en lui-même.

D.2 Téléchargements

Avant toute chose, il est évident que nous devons commencer par télécharger une distribution Linux. Toutes les commandes décrites dans les lignes suivantes ont été testées sur la distribution Linux Mint 15.

Les fichiers du code en lui-même sont hébergés à l'adresse suivante :

`http://sourceforge.net/projects/virtualbelgium/files/`
et sont organisés de la manière suivante :

Listons à présent les différents outils obligatoires à la compilation et/ou exécution du programme.

./	dossier racine, contient les scripts d'exécution ainsi que le "Makefile"
./bin	fichiers d'exécution et de configuration
./data	données d'entrées
./doc	documentation
./include	fichiers d'en-tête
./licenses	licenses de Repast HPC, tinyxml2 et VirtualBelgium
./logs	fichiers log des simulations
./outputs	sorties générées par les simulations
./scripts	scripts pour le traitement des sorties
./src	fichiers sources et le "Makefile"

Compilateur C++

Le projet étant codé en C++, un compilateur de ce langage est nécessaire. Le compilateur le plus connu est *g++*. Il est disponible dans le gestionnaire de dépôts grâce à la commande

apt-get install g++

Outil *Make*

Cet outil permet de simplifier les commandes de compilation lorsque le programme comporte beaucoup de fichiers. Il s'occupe des dépendances et automatise certains processus.

apt-get install g++

La commande Make exécute les règles définies par le fichier Makefile.

L'environnement MPI

Le *Message Passing Interface* (MPI) est un standard permettant à différents processus d'une même exécution du programme de communiquer ensemble. Un utilitaire bien connu est l'exécutable OpenMPI, disponible à nouveau dans le gestionnaire de dépôt par la commande

apt-get install libopenmpi-dev openmpi-common

La librairie Repast HPC

La librairie principale utilisée par notre programme est le framework Repast HPC 1.0.1. Elle permet de modéliser au mieux un système basé sur le c++ et utilisant un grand ensemble d'unités de calcul. À l'origine, elle a été créée par le Laboratoire National d'Argonne. Le code compilable est disponible à l'adresse suivante : `http://repast.sourceforge.net`. Cette librairie en nécessite d'autres avant d'être compilée :

- Curl ;
- Boost 1.48 (ou supérieure) : En plus de cette librairie générale, nous avons besoin des spécifiques suivantes : boost-mpi, boost-system, boost-serialization and boost-filesystem ;
- NetCDF 4.2.1 ;
- NetCDF C++ 4.2.

Les commandes de téléchargement sont simplement :

Pour Curl :

apt-get install libcurl-dev

Pour Boost :

apt-get install libboost-dev libboost-mpi-dev libboost-serialization-dev libboost-system-dev libboost-filesystem-dev

Pour NetCDF :

apt-get install libnetcdf-dev

D.3 Repast HPC

Compilation de Repast HPC

Lorsque tout est téléchargé et extrait de l'archive, la compilation s'effectue grâce aux trois commandes suivantes : (attention à utiliser les privilèges super-utilisateur).

1. ./configure

2. ./make
3. ./make install

Il se peut que la commande "make" génère une erreur. Pour y remédier, il faut remplacer toutes les occurrences de

```
getItems(...)
```

par

```
this->getItems(...)
```

dans les documents

./src/repast_hpc/DirectedVertex.h et
./src/repast_hpc/UndirectedVertex.h

Si une erreur se présente à nouveau lors de l'exécution du "make", le *Makefile* doit être édité de telle façon à supprimer toutes les références vers les modèles *Zombie* et *Rumor*.

D.4 Compilation et exécution de VirtualBelgium

Ordinateur personnel

Pour compiler simplement le programme sur un ordinateur personnel, il suffit d'exécuter la commande

make

Après la compilation, il faut lancer l'exécutable. Un script a été créé spécialement, il s'agit de la commande :

./run.sh NP

où NP est le nombre de processeurs désirés.

Exécution de calculs intensifs

Pour des simulations mettant en scène un nombre important d'agents, il est recommandé d'effectuer les opérations sur un système de plusieurs ordinateurs, "cluster". Suivant les appareils utilisés, il est possible que le fichier "Makefile" doive être modifié. Si le cluster utilisé est *Lemaitre* du *Consortium des Equipements de Calcul Intensif* (`http://www.ceci-hpc.be/`), la compilation se fait simplement avec la commande

make ucl

Ensuite, l'exécution se lance avec le script *run_lemaitre2.sh*

sbatch run_lemaitre2.sh

Ce script permet aussi de personnaliser l'exécution en proposant à l'utilisateur les options suivantes :

— mail-user : une adresse e-mail pour les notifications ;
— time : le temps d'exécution demandé ;
— ntask : le nombre de processeurs voulu ;
— mem-per-cpu :la mémoire réquisitionnée par processeur.

Debuggage

Si le code est modifié pour un ajout de fonctionnalité ou pour une optimisation, il est préférable de pouvoir le débugger si une erreur survient lors de la compilation ou l'exécution. À nouveau la commande "make" va nous être utile en lui ajoutant l'option "debug"

make debug

Cette commande fait appel au débugger GNU gdb dont la documentation peut être trouvée à `http://www.sourceware.org/gdb/documentation/`.

D.5 Configuration de VirtualBelgium

VirtualBelgium est un programme de simulation par agent contenant plusieurs modèles possibles. Le choix du modèle se fait dans le fichier

\bin\model.props

Ce fichier reprend aussi le choix, entre autres, du réseau, de la population synthétique ou encore de la partie à simuler [1].

D.6 Mise à jour du code

Les dernières versions du code de VirtualBelgium sont disponibles sur un répertoire Subversion (SVN) hébergé à l'Université de Namur. Lorsque le logiciel Subversion est installé sur notre machine, les données sont récupérables en trois étapes :

1. Demander un compte à virtualbelgium@math.unamur.be ;
2. Créer un tunnel SSH vers Gauss :

   ssh -N -f -l 5555 :localhost :3690 user@gauss.math.fundp.ac.be

3. Se connecter à

   svn co svn ://user@localhost :5555/virtualbelgium/

   pour recevoir la dernière version du programme

Les différentes commandes de Subversion pour éditer un projet sont listées à `http://svnbook.red-bean.com/index.en.html` (disponible en français).

1. Chaines d'activités, naissances, morts ou âges

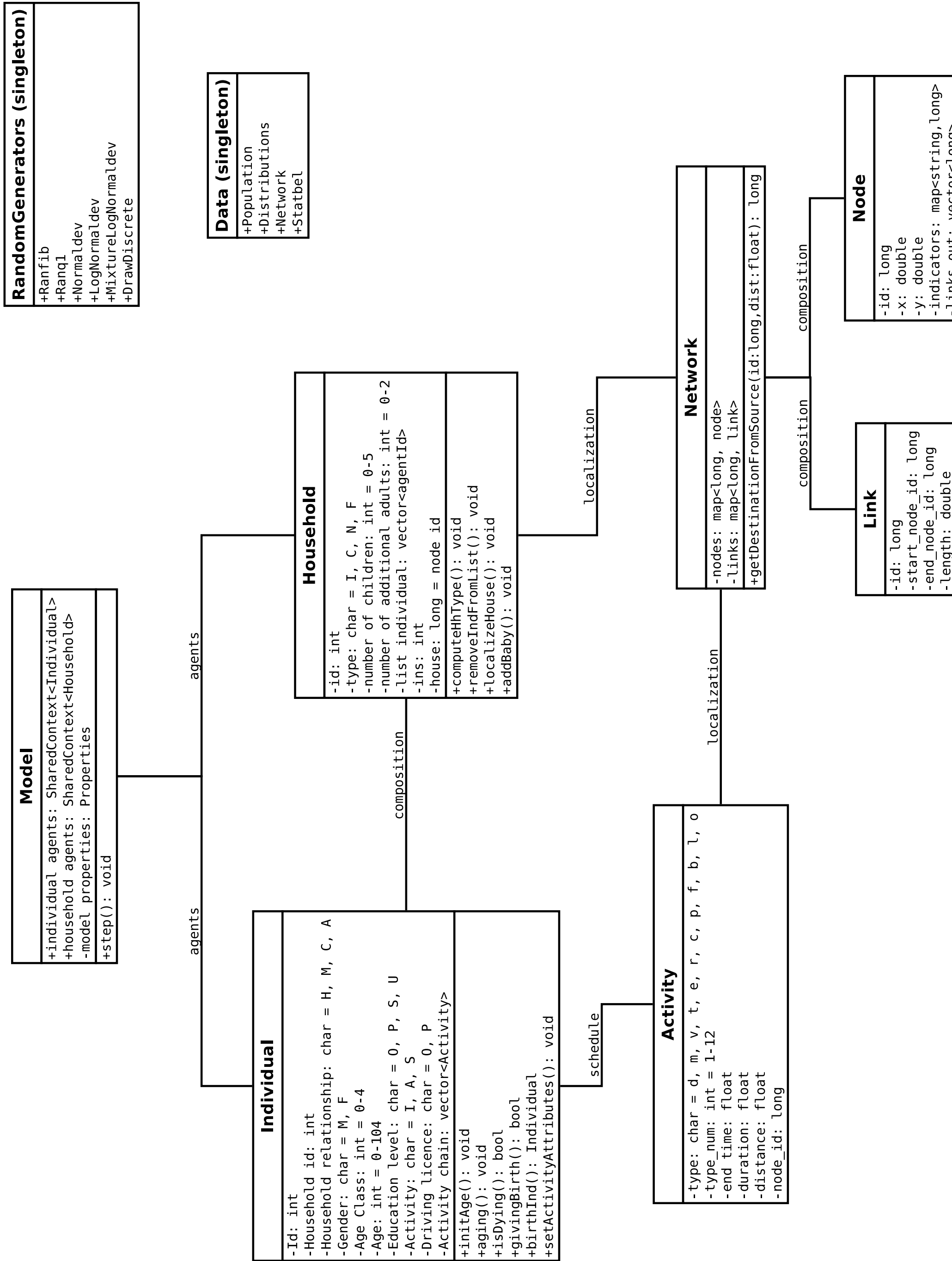

FIGURE D.1 – Diagramme des classes pour VirtualBelgium

Bibliographie

[1] « Agents et systèmes mutliagents » – http://www.damas.ift.ulaval.ca/~coursMAS/ComplementsH10/Agents-SMA.pdf, consulté le 5/11/2013.

[2] « Banque de données terminologiques et linguistiques du gouvernement du canada » – http://www.btb.termiumplus.gc.ca/tpv2alpha/alpha-fra.html?lang=fra&i=&index=alt&__index=alt&srchtxt=satisficing, consulté le 8 août 2014.

[3] « Openstreetmap, la carte coopérative libre » – http://www.openstreetmap.org/, consulté le 13/04/2013.

[4] J. Barthélemy – « Modèles de Choix Discret : Introduction », Introduction faite dans le cadre du cours [SMATM125] de l'université UNamur, 2007.

[5] J. Barthélemy et P. Toint – « Synthetic population generation in presence of data inconsistencies », Décembre 2010, accessible sur http://www.unamur.be/recherche/projets/page_view/11278201/, consulté le 07/05/2013.

[6] J. Barthélemy, P. Toint et E. Cornelis – « A parallelized microsimulation platform for population and mobility behaviour : application to Belgium », Thèse, Unamur : Faculté des sciences, 2014.

[7] F. Billari, A. Prskawetz, B. A. Diaz et T. Fent – « The "Wedding-Ring" : An Agent-Based marriage model based on social interaction », *Demographic Research* **17** (2007), p. 59–82.

[8] D. M. Buss – « Human mate selection : Opposites are sometimes said to attract, but in fact we are likely to marry someone who is similar to us in almost every variable », *American Scientist* (1985), p. 47–51.

[9] B. Chaib-draa, I. Jarras et B. Moulin – *Systèmes multiagents : Principes généraux et applications*, Hermès, 2001.

[10] N. Collier – *Repast HPC Manual*, Février 2012, accessible sur `http://repast.sourceforge.net/docs/RepastHPCManual.pdf`, consulté le 24/04/2013.

[11] Dico Info – `http://dictionnaire.phpmyvisites.net/`, consulté le 5/11/2013.

[12] T. S. Ferguson – « Who solved the secretary problem ? », *Statistical Science* **4** (1989), no. 3, p. 282–289.

[13] A. T. Fiore et J. S. Donath – « Homophily in Online Dating : When Do You Like Someone Like Yourself ? », *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, 2005, p. 1371–1374.

[14] W. Furman et V. A. Simon – « Homophily in adolescent romantic relationships », *Understanding Peer Influence in Children and Adolescents ; Prinstein, MJ ; Dodge, KA, Eds* (2008), p. 203–224.

[15] Futura-sciences : High-tech – `http://www.futura-sciences.com/magazines/high-tech/`, consulté le 17/04/2014.

[16] D. Gale et L. S. Shapley – « College admissions and the stability of marriage », *The American Mathematical Monthly* **69** (1962), no. 1, p. 9–15.

[17] D. Gale et M. Sotomayor – « Some remarks on the stable matching problem », *Discrete Applied Mathematics* **11** (1985), p. 223–232.

[18] D. Gregor et M. Troyer – « Boost.MPI », accessible sur `http://www.boost.org/doc/libs/1_53_0/doc/html/mpi.html`, consulté le 28/04/2013, 2005-2006.

[19] D. Gusfield – « Three fast algorithms for four problems in stable marriage », *SIAM Journal on Computing* **16** (1988), no. 4, p. 742–769.

[20] P. Heymans – « Conception et Programmation Orientées Objet [INFO-B234] », Cours donné à l'université UNamur, 2012-2013.

[21] J.-P. Hubert et P. Toint – « La mobilité quotidienne des Belges », Tech. report, Presses universitaires de Namur, 2002.

[22] Insee : Institut nationalde la statistique et des études économiques – `http://www.insee.fr/fr/themes/tableau.asp?reg_id=0&ref_id=nattef02311`.

[23] N. Jennings, K. Sycara et M. Wooldridge – « A Roadmap of Agent Research and Development », *Autonomous Agents and Mutli-Agent Systems* (1998).

[24] M. Kalmijn – « Intermarriage and homogamy : Causes, patterns, trends », *Annual review of sociology* (1998), p. 395–421.

[25] M. Kalmijn et H. Flap – « Assortative meeting and mating : Unintended consequences of organized settings for partner choices », *Social forces* **79** (2001), no. 4, p. 1289–1312.

[26] J. Kleinberg et E. Tardos – *Algorithm design*, Pearson Addison-Wesley, 2006.

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

[28] R. Kurzban et J. Weeden – « Hurrydate : Mate preferences in action », *Evolution and Human Behavior* **26** (2005), no. 3, p. 227–244.

[29] J. H. Langlois, L. Kalakanis, A. J. Rubenstein, A. Larson, M. Hallam et M. Smoot – « Maxims or myths of beauty ? a meta-analytic and theoretical review. », *Psychological bulletin* **126** (2000), no. 3, p. 390.

[30] D. McVitie et L. Wilson – « Stable marriage assignment for unequal sets », *BIT Numerical Mathematics* **10** (1970), p. 295–309.

[31] — , *The stable marriage problem*, Technical report series, no. 12, University of Newcastle upon Tyne,Department of Agricultural Economics & Food Marketing,Centre for Rural Economy, 1970.

[32] NOTRE BELGIQUE.BE – « La Belgique, ses communes et ses dépendances », `http://www.notrebelgique.be/fr/index.php?nv=1&PHPSESSID=b8b0b83f9415a649fb4cfe4858a35fad`, consulté le 25 mai 2014, mise à jour le 7/11/12, 2008-2012.

[33] J.-F. Rabasse – « Le langage C++ - Introduction à la programmation orientée ob jets, présentation du langage C++ », accessible sur `http://astro.ens.fr/osae/cxx.pdf`, consulté le 08/05/2013.

[34] R. Ramey – « Boost C++ librairies : Serialization », accessible sur `http://www.boost.org/doc/libs/1_53_0/libs/serialization/doc/index.html`, consulté le 28/04/2013, 2002-2004.

[35] P. C. Regan, L. Levin, S. Sprecher, F. S. Christopher et R. Gate – « Partner preferences : What characteristics do men and women desire in their short-term sexual and long-term romantic partners ? », *Journal of Psychology & Human Sexuality* **12** (2000), no. 3, p. 1–21.

[36] S. Scott – « A study of stable marriage problems with ties », Thèse, University of Glasgow, 2005.

[37] Statbel – « Mariages, divorces et cohabitation légale - Statistiques & Analyses », http://statbel.fgov.be/fr/statistiques/chiffres/population/mariage_divorce_cohabitation/, consulté le 11/08/2014.

[38] P. Todd et G. Miller – « From pride and prejudice to persuasion : Satisficing in mate search », Simple heuristics that make us smart, Oxford University Press, New York, 1999, p. 287–308.

[39] K. E. Train – *Discrete Choice Methods with Simulation*, 2 éd., Cambridge University Press, 2009.

[40] Wikipédia – « Logit », http://en.wikipedia.org/wiki/Logit, consulté le 18/08/2014.

[41] — , « Satisficing », http://fr.wikipedia.org/wiki/Satisficing, consulté le 8 août 2014.

[42] Wikipédia : l'encyclopédie libre – http://fr.wikipedia.org/wiki/Wikip%C3%A9dia:Accueil_principal, consulté le 5/11/2013.