MASTER EN SCIENCES MATHÉMATIQUES

Matching on school choice: theory and algorithms

MARELLI, Virginie

Award date:
2013

Awarding institution:
Universite de Namur

MASTER EN MATHÉMATIQUES

Matching on school choice: theory and algorithms

Virginie Marelli

2013

UNIVERSITE DE NAMUR

Faculté des Sciences

MATCHING ON SCHOOL CHOICE : THEORY AND ALGORITHMS

Mémoire présenté pour l'obtention

du grade académique de master en Sciences mathématiques, à finalité approfondie

Virginie MARELLI

Juin 2013

Matching on school choice :

Theory and Algorithms

Virginie MARELLI

Résumé

Dans ce mémoire, nous appliquons les algorithmes génétiques à des problèmes de *matching*. Dans un premier temps, nous passons en revue toute la théorie du *matching*, tant d'un point de vue économique que d'un point de vue algorithmique. Nous analysons trois problèmes de *matching* : le cas simple du *one-to-one matching* ou mariage ; le *many-to-one matching* sans contrainte et le *many-to-one matching* avec contraintes. Dans un deuxième temps, nous décrivons brièvement les algorithmes génétiques, leur mode de fonctionnement et leurs adaptations. Nous les intégrons dans un problème simple de *matching* : le problème du mariage afin de les tester. Ensuite, nous les appliquons à un cas réel de répartition des élèves dans les écoles belges. Ce problème est sujet à des contraintes imposées par le décret "missions". Ce cas de *many-to-one matching* avec contraintes et indifférences n'a pas encore été traité dans la littérature existante et les algorithmes génétiques donnent de bons résultats. Dans un dernier temps, nous appliquons encore les algorithmes génétiques à un autre problème de *matching* : la formation de coalitions. Dans ce cas également, l'apport de ces algorithmes s'avère intéressant.

Abstract

In this master thesis, we apply genetic algorithms to matching problems. The first chapter takes a look at the literature on matching and three famous problems : one-to-one matching (marriage problem) and many-to-one matching (school choice problem) with and without constraints. Then, we briefly describe genetic algorithms and how they work. We apply them to the simple marriage problem to develop and test them, before we look into the real Belgian school choice problem, with indifferences and constraints. This problem has never been considered before and our genetic algorithms yield good results. Finally, we apply GA to the hedonic games, once again obtaining interesting results.

Mémoire de Master en Sciences Mathématiques, à finalité approfondie
Juin 2013
Promoteurs : T. Carletti, G. Aldashev

Highlights

Very often operational optimisation problems do not fit in the "classical" optimisation framework, that requires smooth functions and the possibility to access, speedily and easily, to their derivatives. Actually the former ones usually lack both requirements, functions are defined on discrete variables, sometimes also on categorical ones, and thus derivatives do not exist ; however, even when derivative are computable in principle, the huge number of involved variables or the definition of function through recursive schemes, make the computation of the latter almost impossible.

Some of the above problems, can be solved or their impact to be reduced, using heuristic methods. *Genetic Algorithms*, for short GAs in the following, belong to this class. They are directly inspired by biology and Darwinian evolution : organisms are suited to their habitats (Adaptation) ; offsprings are similar but not equal to their parents (Inheritance) ; new, better adapted types of organisms, emerge and those that fail to change adequately are subject to extinction (Natural selection). Thus the fittest individuals have a high chance of having a large number of offsprings and finally their traits can spread across the population, generation by generation.

The aim of this master thesis is to apply GAs on matching problems, whithin the *hedonic games* and the *students-school assignment*. To the best of our knowledge this is the first time that GAs are applied, in this way, to matching problems, the literature being very scarce [AC99].

Matching is a well known problem and relevant to many field of economics research ; the available literature provides nice theoretical results, but whose operational implementation is often by far unreachable, because direct application of such results will produce algorithm whose time complexity will be larger than polynomial. Let us finally observe that because of the combinatoric nature of the problems, brute force enumeration of all possible cases becomes rapidly impossible even for small sizes problems.

We consider the problem of matching in school choice with constraints and indifferences. Assigning students to schools has never been easy since there is always someone to complain. Giving every student his preferred school is not always possible because of the limited schools' capacity. Moreover, some well-praised schools are overwhelmed by students proposals and cannot deal with it. That is why it is common to use a central house to avoid congestion and to try to satisfy students as well as schools. They usually use the Deferred Acceptance Algorithm (DAA) [Ald10] which yields a stable matching that is optimal for students. Here, stable means that no-one has incentive to break the deal (the inscription). Furthermore, this direct mechanism is strategy-proof for students since the dominant strategy for students is to reveal their true preferences.

Since every school chooses the way it ranks students, there exists some inequalities among students. To solve this problem, states introduce quotas to fight again social segregation or economic segregation [Abd05, RS90]. Every student gets a type, for example race, social status, and quotas are introduced to regulate the number of students of different types that should be present in each school. Let us observe that either quotas can be strictly respected (hard bounds approach) or they should serve more as guidelines and thus consider students' preferences firstly. Another source of complexity arises when schools can not strictly rank all students and a tie-breaking rule is needed in order to use the DAA. As a consequence of this random sorting, the DAA does not longer necessarily find an optimal matching. This loss of utility raises the following questions : can efficiency be restored ? Is there a mechanism that Pareto-improve the results obtained using existing algorithms ? Our goal is to positively answer to these questions

by introducing a new method based on GAs to this problem, more precisely we will show that the efficiency can be partly restored and the solution Pareto-improved.

The second matching problem that we analyze is about the formation of coalitions and in particular the renowned *hedonic games*.

A finite set of players must be divided according to their preferences, into disjoint groups, called coalitions, that should partition all the individuals. In the following we will focus on anonymous games with single peaked preferences, meaning that players only care about the size of the coalition or the amount of public good it produces. A partition is stable if there is no envy among the players, let us observe that stability is a hard condition to be satisfied and hence we must often rely on a relaxed weaker notion, to be able to have positive results. In this setting, Bogomolnaia et al. propose an algorithm which gives a stable and weakly Pareto-optimal partition under some suitable conditions. Weakly Pareto-optimal means that there exists no partition that all players strictly prefer. If these assumptions are not met, the algorithm can only find a stable partition. As already stated most of the existing algorithms are not totally satisfying, we thus hereby introduce a new method based on GAs that allows us to Pareto-improve the result by Bogomolnaia et al. Such a Pareto-improvement is very interesting because some players are better off, while the remaining ones are left with their previous assignment.

Through this thesis, we apply mathematical processes to economic problems and show the importance of understanding both fields.

Remerciements

Je tiens particulièrement à remercier Timoteo Carletti pour son soutien, sa patience et ses encouragements tout au long de l'élaboration de ce mémoire. Plus qu'un professeur, il a su me conforter dans mes projets.

Je tiens également à remercier Gani Aldashev, qui a soutenu mon initiative d'appliquer les mathématiques à un problème de microéconomie, et m'a permis de rencontrer Antonio Nicolò.

Je remercie ce dernier pour son accueil chaleureux à Padoue, son enthousiasme et sa confiance en ma recherche. J'ai beaucoup apprécié travailler avec Antonio, il a permis à mon projet de master en économie de se réaliser.

Ces trois personnes ont éveillé en moi un intérêt nouveau pour la recherche. Dans ce mémoire, j'ai pu traiter un sujet qui me tient spécialement à coeur, à la frontière entre les mathématiques et l'économie.

Je remercie aussi tous les professeurs du département de mathématiques de Namur, qui m'ont permis de m'épanouir durant ces cinq années d'étude dans un climat familial. En particulier, merci à Anne Lemaître qui a toujours été présente pour me soutenir et m'écouter.

Je ne peux pas oublier les étudiants de mathématiques, notamment ceux de ma classe, avec lesquels j'ai passé de bons moments qui resteront gravés dans ma mémoire.

Je remercie mon *teammate* d'avoir relu mon mémoire, supporté mes indécisions avec tant de calme et de m'avoir conforté jusqu'au bout.

J'adresse ce dernier remerciement, mais non le moindre, à ma famille, sans qui je ne serais pas aussi loin maintenant. Elle m'a apporté son soutien précieux et m'a toujours encouragé à me dépasser. Une petite pensée pour mon frère François, qui a supporté sa grande soeur sans jamais broncher.

Que les personnes sans qui je n'aurais pas pu accomplir ce mémoire, et que je n'ai pas citées ici, me pardonnent.

Virginie Marelli

Table des matières

Chapitre 1

Introduction

"Mathematics is a game played according to certain rules with meaningless marks on paper. "
David Hilbert

Les mathématiques sont redoutées par beaucoup, elles provoquent des angoisses, donnent des sueurs froides et font faire des cauchemars. Pourtant, quand on y regarde d'un peu plus près, elles sont partout, le matin dans le radio-réveil, dans la machine à café, dans le réseau routier et même dans l'ascenseur qui vous conduit au bureau. Leur domaine d'application est vaste, et dans ce mémoire c'est à la microéconomie que nous allons les appliquer. Plus particulièrement, nous avons choisi d'étudier la théorie du *matching*, une branche à la frontière entre les mathématiques et l'économie.

Nous avions besoin d'un problème, la Belgique nous en a donné un, avec son décret inscription dans les écoles secondaires. Pas toujours bien compris et par ailleurs très controversé, ce décret donne les règles d'acceptation des élèves dans les écoles. En quelques mots, les élèves sortant de primaire font une demande d'inscription dans l'école secondaire de leur choix et sont acceptés ou non, selon le nombre de places disponibles et des critères d'acceptation tels que la distance de l'école au domicile de l'élève, son statut économique (défavorisé ou non), etc. Les élèves qui ne sont pas acceptés dans leur première école, sont redirigés vers l'école de leur second choix, voire troisième ou quatrième choix selon que les écoles sont complètes ou non. Dans ces écoles, ils sont encore soumis à la procédure d'acceptation selon les mêmes critères. En plus de ceux-ci, les écoles doivent réserver au moins 20% de leurs places pour les élèves défavorisés. Ce décret vise à la mixité sociale dans les écoles, en donnant à chacun sa chance mais la décentralisation des demandes d'inscription entraine beaucoup de problèmes. Chaque année, depuis l'entrée en vigueur de cette loi, il y a toujours des élèves sans école le premier septembre et déjà, pour l'année scolaire 2013-2014, 1186 élèves sont sur liste d'attente, faisant rager autant de parents (voir Le Soir, [Gof13]). Cette situation n'est autre qu'un problème de *matching* et une petite explication s'impose.

La traduction littérale : *correspondance* définit déjà en elle-même le *matching*. Un *matching* est défini comme une application qui associe entre eux des éléments d'au moins deux ensembles différents, suite à des préférences individuelles. C'est une branche de la théorie des jeux, utilisée dans beaucoup de domaines, par exemple dans les agences matrimoniales. Dans ce cas, on parlera de *one-to-one matching* car il s'agit de faire correspondre une personne avec une autre du sexe opposé. Chaque individu (hommes et femmes) donne ses préférences sur les individus de l'autre sexe (liste de préférences) et le but est de trouver un *matching* qui maximise le bien-être des deux parties (que chacun ait un partenaire haut placé dans sa liste de préférences). On dit d'un *matching* qu'il est stable s'il n'existe pas de couple qui aurait préféré être ensemble,

autrement dit si personne n'a intérêt à rompre son contrat de mariage. On parle de *strategy-proofness* lorsque tous les participants ont intérêt à révéler leurs vraies préférences sur les individus de l'autre sexe [1].

La théorie du *matching* est plutôt d'actualité et d'ailleurs, Shapley et Roth ont reçu le prix Nobel d'économie en 2012 pour leurs travaux sur le *matching* et la microéconomie.

Lorsqu'on passe au *many-to-one matching*, on met en relation plusieurs individus d'un même ensemble avec une personne d'un autre ensemble. C'est exactement le cas des élèves et des écoles en Belgique puisqu'une école peut accepter un certain nombre d'élèves, dans la limite de ses capacités. Ce problème ne concerne pas seulement les écoles belges mais aussi bien d'autres systèmes scolaires dans le monde. Attribuer une école à un élève n'a jamais été chose facile et il y a toujours des parties lésées, que ce soient les élèves qui n'ont pas l'école de leur choix ou inversement. Aux USA, beaucoup d'écoles et universités utilisent déjà une procédure centralisée, avec une *central house* qui gère toutes les demandes d'inscriptions et attribue à chaque élève une école en essayant de respecter au mieux les préférences de ces derniers. Différents mécanismes [2] ont été mis au point théoriquement et utilisés dans différentes institutions, le plus connu est celui développé par Gale et Shapley en 1962 : le *Deferred Acceptance Algorithm* (DAA) [3]. A la première étape, les élèves font une demande à l'école de leur premier choix, et dans chaque école sont assignés provisoirement les élèves que celle-ci préfère, en respectant la capacité maximale ; les autres élèves sont rejetés. A l'étape suivante, les élèves rejetés font une demande à l'école de leur second choix et sont assignés provisoirement en suivant le même procédé qu'à l'étape une. Et ainsi de suite jusqu'à ce que tous les élèves soient inscrits dans une école ou qu'ils aient été rejetés par toutes les écoles de leur liste de préférences. Cet algorithme se termine en un temps fini et donne un *matching* stable et optimal pour les étudiants. Stable signifie qu'aucun individu n'a envie de rompre le contrat, dans ce cas l'inscription ; et optimal signifie que c'est le *matching* préféré par tous les étudiants. Cependant, dans notre problème belge, nous avons une donnée supplémentaire : les quotas.

Les quotas ont été introduits afin de contrer la ségrégation sociale ou économique et d'éviter d'avoir des écoles ghettos. En effet, avant l'introduction d'une loi régulant les inscriptions, tous les élèves n'étaient pas égaux dans les écoles ; ces dernières décidaient elles-mêmes de leurs critères d'acceptation des élèves et les classaient comme bon leur semblait. Pour pallier à ce problème, l'Etat ou le pouvoir en place définit des 'types' d'élèves et à partir de ceux-ci, des quotas pour chaque école. Dans notre cas, nous avons 2 types (défavorisé ou non) et chaque école réserve au moins 20% des places disponibles pour les élèves défavorisés [4]. Il existe deux approches développées par Elhers et al. dans [EHYY12] : une première approche qui consiste à respecter strictement les quotas (*hard bounds*) et une seconde qui fait passer les préférences des élèves avant les quotas (*soft bounds*). Nous détaillerons plus avant ces 2 approches.

Lorsque les écoles ne savent pas classer strictement les élèves (préférences non-strictes), les algorithmes existants, que ce soit avec ou sans contrainte (quotas), rencontrent certains problèmes. En effet, ces algorithmes ne fonctionnent qu'avec des préférences strictes, il faut donc introduire une règle (*tie-breaking rule*) pour départager les élèves. Abdulkadiroğlu, Pathak et Roth, dans [APR09] ont étudié ce problème pour le cas sans contrainte, et en ont conclu une perte d'efficience des algorithmes dans ces circonstances : les élèves ne sont plus assignés à la meilleure école possible (parmi les *matching* existants). Erdin et Ergil ont quant à eux

1. Ce problème est pour la première fois introduit par Gale et Shapley dans [GS62] et est synthétisé par Roth et Sotomayor dans [RS90].
2. Voir Abdulkadiroğlu et Sönmez, dans [AS03].
3. Voir Gale and Shapley, dans [GS62].
4. Voir le texte du décret, [Decret11].

tenté de retrouver l'efficience, dans leur article [EE08]. Cependant, à notre connaissance il n'y a aucun papier qui traite à la fois des préférences non-strictes et des contraintes dans les écoles. Dès lors, nous nous sommes penchés sur ce problème et avons choisi pour améliorer les algorithmes existants d'utiliser les algorithmes génétiques.

Un algorithme génétique est une méthode d'optimisation heuristique qui explore l'espace des solutions. Il n'a pas besoin des dérivées de la fonction à optimiser, il peut ainsi aussi bien optimiser des fonctions discrètes que continues et n'est pas ralenti par des erreurs d'arrondi. Comme son nom l'indique, cet algorithme est basé sur les principes de la génétique et mime ses techniques. Chaque solution est un individu et se voit attribuer une *fitness* plus ou moins élevée selon l'adaptation de l'individu à son milieu (un individu adapté aura une *fitness* élevée). Le but de l'algorithme génétique est de faire évoluer une population initiale d'individus. Ainsi, l'algorithme sélectionne les meilleurs individus de la population initiale, ceux qui ont la *fitness* la plus élevée. Ensuite vient une phase de reproduction de ces individus, par croisement des solutions, et enfin une phase de mutation qui assure la diversité de la population[5]. Ces trois étapes sont répétées jusqu'à l'obtention de la condition d'arrêt. Le seul bémol est que les algorithmes génétiques sont heuristiques et la convergence n'est pas certaine. En théorie des jeux, les algorithmes génétiques sont généralement utilisés pour modéliser le comportement des joueurs : ceux-ci utilisent les étapes précédentes du jeu pour adapter leur comportement afin d'améliorer leur résultats[6]. Les algorithmes génétiques n'ont pour ainsi dire pas été utilisés dans un but purement d'optimisation dans la théorie du *matching*, et c'est ce que nous avons fait dans ce mémoire.

La théorie du *matching* ne se limite pas simplement aux écoles et aux élèves, nous avons également abordé un problème d'un autre genre : la formation de coalitions. Cela consiste à partitionner un groupe de joueurs en coalitions (groupes disjoints). Cependant, comme le nombre de coalitions augmente exponentiellement par rapport au nombre de joueurs, il est très difficile d'avoir des résultats satisfaisants. Nous nous sommes donc restreints au cas des jeux hédoniques, où les joueurs se soucient seulement de la coalition dans laquelle ils se trouvent. Là encore, les résultats positifs font défaut et il faut restreindre le domaine des préférences des joueurs aux préférences anonymes et *single-peaked*. Les joueurs ne sont donc intéressés que par la taille de leur coalition, qu'ils désirent optimale : le pic, plus on s'en rapproche et plus les joueurs sont contents. Dans ce cas, Bogomolnaia et Jackson dans [BJ02], obtiennent une partition stable et optimale sous certaines conditions. Néanmoins, lorsque les conditions d'existence ne sont pas satisfaites, l'algorithme de Bogomolnaia et al. ne donne pas une partition optimale et c'est là que les algorithmes génétiques interviennent pour essayer de retrouver l'efficience perdue.

La théorie du *matching* est plutôt bien développée et la première partie de ce mémoire vise à survoler la littérature sur le *matching*, afin de bien comprendre les enjeux tant économiques qu'algorithmiques. Ce chapitre permettra au lecteur d'avoir un aperçu de la théorie du *matching* pour comprendre la suite du travail. Le chapitre 3 est consacré aux algorithmes génétiques et à la description d'un algorithme génétique de base. Dans le chapitre 4, nous essayons les algorithmes génétiques sur le problème simple du mariage (*one-to-one matching*) afin d'intégrer les techniques. Le chapitre 5 est entièrement dédié au cas belge et à l'application des algorithmes génétiques au problème du *many-to-one matching*. Nous envisageons deux approches, l'une plus mathématique : optimisation des différents critères du décret belge ; l'autre plus économique : amélioration des algorithmes déjà existants dans la théorie du *mat-*

5. Les algorithmes génétiques sont introduits par Goldberg, dans [Gol89]
6. Voir Ünver [Unv05] et Haruvy et al. [HRU06]

ching. Nous récupérons partiellement la perte d'efficience causée par l'indifférence des écoles. Cette approche a été spécialement travaillée durant le stage de recherche à Padoue, ainsi que le chapitre suivant. Dans ce dernier, nous avons appliqué les algorithmes génétiques aux jeux hédoniques. Encore une fois, le but était d'améliorer les algorithmes existants, et l'objectif a été atteint. Nous avons d'ailleurs écrit un papier (en phase de soumission) en collaboration avec les professeurs Carletti et Nicolò, qui expose nos résultats. Enfin, nous terminons ce mémoire par une conclusion du travail effectué et les perspectives pour la recherche future.

Les codes utilisés dans ce mémoire ont été développés par l'auteur et en annexe se trouve un mode d'emploi simplifié de ceux-ci.

Chapitre 2

Résumé de la littérature sur le *matching*

"Economics has somehow a reputation among some part of the public as being boring (⋯), but I've always thought of Economics as being as much as part of the social sciences as part of the humanities because it gives us a window into people's lifes and some of the biggest crossing points, when they are getting to schools and universities, when they get jobs, when they choose carreers, when they get married, these are all matching markets. "
Alvin Roth, lors de la remise du prix Nobel

2.1 *One-to-one matching*

L'exemple le plus courant de la théorie du *matching* est le problème du mariage, qui est équivalent à celui que rencontrent les agences matrimoniales. Considérons un ensemble d'hommes et de femmes qui recherchent l'âme soeur. Grâce à la théorie du *one-to-one matching*, nous espérons répondre à leurs attentes et leur trouver le partenaire idéal. Le principe est très simple, pour commencer, chaque individu classe les personnes de l'autre sexe par ordre de préférences décroissantes. Si un individu préfère rester célibataire plutôt que de se voir attribuer une personne qui ne lui convient pas, il ne placera pas le nom de cette ou ces personnes sur sa liste et terminera celle-ci par son propre nom. Nous poserons comme hypothèse que les préférences des individus sont réflexives et transitives ainsi que strictes, afin d'éviter les ex-aequo. Nous supposons également que les préférences des participants sont indépendantes de celles des autres membres du groupe.
L'objectif final est de former des couples homme-femme, en respectant les préférences de chacun. Il est évident que, pour qu'une personne soit assignée à une autre, il faut qu'elle soit sur la liste de celle-ci sinon le couple ne pourra pas se former. On dira qu'un individu ne figurant pas sur la liste de préférences d'une personnes est "non-acceptable" pour cette personne. Avant d'entrer plus avant dans les détails, passons à la modélisation.

Le modèle du *one-to-one matching* consiste en

1. Deux ensembles finis de n et m agents : $M = \{m_1, m_2, ..., m_n\}$ et $W = \{w_1, w_2, ..., w_m\}$

2. Les préférences strictes des deux parties : $P_M = \{P_{m_1}, P_{m_2}, ..., P_{m_n}\}$,
   $w_i P_{m_j}\ w_k$ signifie que m_j préfère w_i à w_k
   et $P_W = \{P_{w_1}, P_{w_2}, ..., P_{w_m}\}$,
   $m_i P_{w_j}\ m_k$ signifie que w_j préfère m_i à m_k

Le modèle du *one-to-one matching* se résume en

$$(M, W, P_M, P_W)$$

A partir de ce modèle, définissons les concepts de base.

Définition 2.1.1. *Matching*

Un matching μ est une correspondance, de l'ensemble $M \cup W$ dans lui-même, telle que

- $\mu(m) \neq m$, *alors* $\mu(m) \in W$
- $\mu(w) \neq w$, *alors* $\mu(w) \in M$
- $\mu(m) = w \Longleftrightarrow \mu(w) = m$

Cette définition est proposée par Alvin Roth et Marilda Sotomayor dans [RS90].

Stabilité

Une question se pose à présent : lorsque tous les couples sont formés, sont-ils heureux ? Vont-ils rester toute leur vie ensemble, ou finiront-ils par divorcer ? Existe-t-il des *matching* qui rendent tous les couples heureux ? La stabilité du *matching* dépend de la satisfaction des couples formés.

Définition 2.1.2. *Paire qui bloque le matching*

Supposons qu'il existe un homme m et une femme w qui ne sont pas ensembles, sous le matching μ, mais qui se préfèrent mutuellement à leur partenaire assigné. C'est-à-dire, $mP_w\,\mu(w)$ (w préfère m à son partenaire) et $wP_m\,\mu(m)$ (m préfère w à son partenaire), dans ce cas, on dira que cette paire (m, w) bloque le matching car ils seraient plus heureux s'ils étaient ensembles.

Définition 2.1.3. *Matching Stable*

Un matching μ est stable s'il n'existe aucune paire qui bloque celui-ci.

Ces définitions sont également proposées par Alvin Roth et Marilda Sotomayor dans [RS90].

Il ne s'agit pas seulement d'avoir un *matching stable*, il faudrait également que celui-ci apporte le plus de satisfaction possible aux individus. L'utilité d'un consommateur est un concept très souvent utilisé en économie et mesure le bien-être de cet individu.

Définition 2.1.4. *Utilité*

L'utilité est une fonction $u : X \subset I\!R^n \to I\!R$ *qui représente une relation de préférence* P*, avec*

$$\forall x, y \in X \quad tq \quad xPy \Leftrightarrow u(x) > u(y)$$

Le but de la fonction d'utilité est de transformer une relation ordinale en relation cardinale, qui est en général plus facile à manipuler. Dans notre modéle, chaque personne possède donc une fonction d'utilité.

Soit la fonction d'utilité d'une femme $w \in W$,

$$\begin{aligned} u_w : \quad & M \to \mathbb{R} \\ & m \rightsquigarrow u_w(m) \end{aligned}$$

Avec, $mP_wm' \Longrightarrow u_w(m) > u_w(m')$, plus l'homme possède un rang élevé dans la liste de w, plus l'utilité de celle-ci sera élevée.

Soit la fonction d'utilité d'un homme $m \in M$,

$$\begin{aligned} u_m : \quad & W \to \mathbb{R} \\ & w \rightsquigarrow u_m(w) \end{aligned}$$

Avec, $wP_mw' \Longrightarrow u_m(w) > u_m(w')$, plus la femme possède un rang élevé dans la liste de m, plus l'utilité de celui-ci sera élevée.

Par exemple, après un *matching* μ, nous aurons donc que l'utilité des participants sera, $\forall w \in W, u_w(\mu(w))$ et $\forall m \in M, u_m(\mu(m))$.

Un individu m préfère un *matching* μ à un *matching* μ', si son utilité est plus élevée sous μ ; en d'autres mots, si $\mu(m)P_m\mu'(m)$, ou encore $u_m(\mu(m)) > u_m(\mu'(m))$.

Un individu m est indifférent tant à un *matching* μ qu'à un *matching* μ', si son utilité est égale pour les deux *matching*. Comme nous avons imposé les préférences strictes, cela se traduit par : $\mu(m) = \mu'(m)$, ce qui entraine $u_m(\mu(m)) = u_m(\mu'(m))$.

Remarque :

Par la suite, nous utiliserons simplement la relation de préférence, sachant qu'elle est équivalente à celle d'utilité.

Définition 2.1.5. *Matching Pareto-optimal*

Un matching stable μ *est Pareto-optimal si tous les participants ont une utilité au moins aussi élevée que dans n'importe quel autre matching.*

Cette définition est donnée par Gale et Shapley dans [GS62].

Deferred Acceptance Algorithm

A présent, nous pouvons introduire le célèbre algorithme *Deferred Acceptance* de Gale et Shapley tiré de [RS90], qui prouve qu'il existe toujours un *matching* stable pour n'importe quel problème de mariage.

Théorème 2.1.1. *Gale and Shapley*

Il existe un matching stable pour n'importe quel problème de mariage.

Démonstration.

Pour démontrer ce théorème, nous utilisons l'algorithme *man-proposing deferred acceptance* (car les hommes proposent).

1. Premièrement, chaque homme fait une proposition de mariage à la femme qu'il préfère. Les femmes qui reçoivent une proposition ne gardent que l'homme qu'elles préfèrent (parmi ceux acceptables) et rejettent les autres. Cependant, ce n'est pas leur choix définitif, elles gardent cet homme en réserve, dans le cas où elles n'auraient pas de meilleure proposition.
   A la fin de cette étape, on obtient un *matching* provisoire μ_1.
2. Deuxièmement, les hommes qui ont été refusés proposent à leur second choix. Là encore, les femmes choisissent leur favori entre les nouvelles propositions et, si elles en ont un, leur homme en réserve. C'est-à-dire, si $mP_w\,\mu_1(w)$ (w préfère m à son partenaire actuel), alors, w choisit m et rejette $\mu_1(w)$, sinon si $\mu(w)P_w\,m$, w garde $\mu_1(w)$ et rejette m. A la fin de chaque étape, chaque femme ne peut garder qu'un seul homme en réserve. On obtient un *matching* provisoire μ_2.
3. Les étapes se poursuivent de la même manière, les hommes rejetés proposent au choix suivant de leur liste et à chaque étape i, on obtient un *matching* provisoire μ_i.
4. L'algorithme s'arrête lorsque chaque homme est en couple avec une femme, ou a été rejeté par toutes les femmes de sa liste. Il reste alors célibataire, tout comme les femmes qui n'ont pas reçu de proposition acceptable.

Cet algorithme produit toujours un *matching* stable, qui est le *matching* final, μ. En effet, supposons par l'absurde qu'il existe une paire (m, w) qui bloque le *matching*.
Dans ce cas, w doit être acceptable pour m et comme $wP_m\,\mu(m)$ (m préfère w à sa partenaire actuelle), m a dû proposer à w avant de proposer à $\mu(m)$ et si $mP_w\,\mu(w)$ (w préfère m à son partenaire actuel) alors, w aurait dû choisir m plutôt que $\mu(w)$. Cette paire (m, w) est en contradiction avec la construction de l'algorithme.
Ce qui prouve bien qu'on a toujours au moins un *matching* stable.

□

Il existe une autre version de cet algorithme dans laquelle ce sont les femmes qui proposent aux hommes et les hommes qui choisissent. Cette version est appelée *woman-proposing deferred acceptance algorithm.*

En plus d'être stable, le *matching* construit par l'algorithme de *man-proposing deferred acceptance* est également optimal pour les hommes (parmi les *matching* stables) : c'est le

matching préféré par tous les hommes, dans l'ensemble des *matching* stables et en même temps, c'est le pire *matching* stable pour les femmes. Inversement, lorsqu'on utilise la version de l'algorithme *woman-proposing*, le *matching* résultant est optimal pour les femmes et c'est le pire *matching* stable pour les hommes. Ces résultats sont donnés par Alvin Roth dans [Rot85].

Dans le cas où les préférences ne sont pas strictes, il faut ajouter un tirage aléatoire qui brise l'ex aequo et ensuite, on peut utiliser le *deferred acceptance algorithm* (DAA), nous en reparlerons dans le Chapitre 5.

Pour mieux comprendre comment fonctionne cet algorithme, voici un exemple avec 3 femmes et 4 hommes.

Exemple 2.1.1.

1. $M = \{m_1, m_2, m_3, m_4\}$
2. $W = \{w_1, w_2, w_3\}$
3. Voici le tableau des préférences des hommes :

P_{m_1}	P_{m_2}	P_{m_3}	P_{m_4}
w_1	w_2	w_1	w_3
w_2	w_1	w_3	w_1
w_3	m_2	w_2	w_2

TABLE 2.1 – Exemple 2.1.1, préférences des hommes

4. Voici le tableau des préférences des femmes :

P_{w_1}	P_{w_2}	P_{w_3}
m_1	m_1	m_3
m_2	m_3	m_2
m_3	m_4	m_4
m_4	m_2	m_1

TABLE 2.2 – Exemple 2.1.1, préférences des femmes

On applique le *man-proposing deferred acceptance algorithm*.

Premièrement, tous les hommes font une proposition de mariage à leur premier choix et comme w_1 reçoit 2 propositions, elle choisit l'homme qu'elle préfère : m_1 et rejette m_3 :

$$\begin{array}{cccccc} \mu_1 & m_1 & m_2 & m_3 & m_4 \\ & w_1 & w_2 & m_3 & w_3 \end{array}$$

Deuxièmement, m_3 propose à son second choix w_3. Celle-ci préfère m_3 à m_4 et rejette donc ce dernier :

$$\begin{array}{cccccc} \mu_2 & m_1 & m_2 & m_3 & m_4 \\ & w_1 & w_2 & w_3 & m_4 \end{array}$$

Troisièmement, m_4 propose à son second choix qui est w_1 mais elle est déjà avec l'homme qu'elle préfère donc m_4 ne peut que proposer à son ultime alternative w_2 qui le préfère à m_2 :

$$\begin{array}{cccccc} \mu_3 & m_1 & m_2 & m_3 & m_4 \\ & w_1 & m_2 & w_3 & w_2 \end{array}$$

Quatrièmement, m_2 propose à w_1 mais m_1 est déjà son homme préféré donc elle le rejette. Il ne reste à m_2 que le choix de célibataire puisqu'il a épuisé sa liste de préférences.

L'algorithme s'arrête car tous les hommes sont engagés avec une femme, sauf m_2 qui a été rejeté par tous ses choix, et le matching final est :

$$\begin{array}{cccccc} \mu^* & m_1 & m_2 & m_3 & m_4 \\ & w_1 & m_2 & w_3 & w_2 \end{array}$$

Strategy-proofness

A présent, mettons-nous à la place des individus : quelle est la stratégie à adopter ? Comment organiser sa liste de préférences ? Imaginons un individu amoureux d'une personne qui ne l'aime pas, va-t-il tout de même rester fidèle à ses préférences, ou bien éliminer cette personne de sa liste afin d'avoir plus de chance de finir en couple avec une personne aimée ? Le fait de révéler ses vraies préférences est-il toujours dans le meilleur intérêt des participants ? Ce genre de réflexion nous amène à la définition de *Strategy-proofness.*

Définition 2.1.6. *Stratégie, best-response et stratégie dominante*

Une stratégie s est un choix que chaque participant peut faire, à n'importe quel moment du jeu.
Nous notons une stratégie $s \in S$, l'ensemble des stratégies.

Soit un individu j, si les stratégies de tous les autres participants sont données par $(s_1^*, s_2^*, ..., s_{j-1}^*, s_{j+1}^*, ..., s_m^*)$,
la meilleure réponse pour j est la stratégie s_j^* qui lui donne le plus d'utilité :

$$u(s_1^*, s_2^*, ..., s_{j-1}^*, s_j^*, s_{j+1}^*, ..., s_m^*) \geq u(s_1^*, s_2^*, ..., s_{j-1}^*, s_j, s_{j+1}^*, ..., s_m^*), \forall s_j \in S_j$$

où S_j est l'ensemble des stratégies disponibles pour j.

Une stratégie s_j^* est dominante pour un agent j si elle est la meilleure réponse à toutes les stratégies possibles des autres agents :

$$u(s_1, s_2, ..., s_{j-1}, s_j^*, s_{j+1}, ..., s_m) \geq u(s_1, s_2, ..., s_{j-1}, s_j, s_{j+1}, ..., s_m), \forall s_i \in S_i, \forall i \in \{1..m\}$$

Définition 2.1.7. *Matching Strategy-proof*

Un matching est strategy-proof si la stratégie dominante de chaque participant est de révéler ses vraies préférences.

Remarque :

Le fait de révéler ses vraies préférences signifie ne pas tricher quant à ses partenaires préférés et les inscrire sur sa liste, mais ne signifie en aucun cas que les préférences sont rendues publiques, elles restent connues du seul participant.

Malheureusement, le théorème suivant affirme qu'un *matching* ne peut être à la fois stable et *strategy-proof*. La définition ci-dessus et le théorème suivant sont tirés de [RS90].

Théorème 2.1.2. *Impossibility theorem (Roth)*

Il n'existe aucun matching stable pour lequel révéler ses vraies préférences est la stratégie dominante de chaque agent.

Démonstration.

Il suffit de prouver que dans n'importe quel problème de mariage, il n'existe aucun *matching* stable et *strategy-proof*.

Prenons le problème à 2 dimensions : un ensemble de 2 femmes et un autre de 2 hommes, avec les préférences suivantes :

P_{w_1}	P_{w_2}	P_{m_1}	P_{m_2}
m_2	m_1	w_1	w_2
m_1	m_2	w_2	w_1

TABLE 2.3 – Théorème 2.1.2, Préférences des hommes et des femmes

Alors, il n'existe que deux *matching* stables, μ et ν. Avec $\mu(m_i) = w_i, \forall i \in \{1,2\}$ et $\nu(m_i) = w_j, \forall i,j \in \{1,2\}, i \neq j$, μ est le *matching* optimal pour les hommes et ν pour les femmes.

Supposons que le *matching* μ est choisi, on observe que si w_2 change ses préférences et supprime m_2 de sa liste, les préférences des femmes deviennent :

P_{w_1}	P_{w_2}
m_2	m_1
m_1	w_2

TABLE 2.4 – Théorème 2.1.2, Préférences modifiées des femmes

Dans ce cas, le seul *matching* réalisable est ν, qui est favorable à w_2 puisque c'est le *matching* optimal pour les femmes.

Inversement, si c'est le *matching* ν qui est choisi, alors m_1 enlève w_2 de ses préférences. Les préférences sont décrites à la page suivante.

P_{m_1}	P_{m_2}
w_1	w_2
m_1	w_1

TABLE 2.5 – Théorème 2.1.2, Préférences modifiées des hommes

Alors, le seul *matching* réalisable est μ, le *matching* optimal pour les hommes.

On vient de prouver qu'il n'existe aucun *matching* stable qui soit *strategy-proof*. Le fait de révéler ses vraies préférences n'est donc pas une stratégie dominante pour tous les agents.

□

Cependant, Roth (dans [Rot85]) démontre que si un mécanisme donne toujours le *matching* optimal pour les hommes (*femmes*), alors leur stratégie dominante est de révéler leurs vraies préférences (mais ce n'est pas celle des femmes (*hommes*)).

La *strategy-proofness* est une propriété importante car sans elle, les joueurs peuvent manipuler leurs préférences et tricher pour obtenir un meilleur partenaire.

Cette théorie est appliquée dans bien d'autres cas que celui du mariage, comme dans le cas des dortoirs scolaires, ou encore dans le domaine médical pour les dons d'organes ou de cellules. En effet, il faut que les groupes sanguins et rhésus, du donneur et du patient, correspondent pour qu'une greffe ne soit pas rejettée par le patient. Vous trouverez également plus de résultats sur la stabilité et la *strategy-proofness* dans la littérature [1], mais nous ne nous attarderons pas car ce qui nous intéresse, c'est le *many-to-one matching*. A présent que la base de la théorie est plantée, nous pouvons passer à la seconde partie.

2.2 *Many-to-one matching : School choice*

Le problème est apparu pour la première fois aux USA avec les stages de médecine. Après leurs années d'université, les étudiants doivent trouver un stage en tant qu'interne dans un hôpital. D'un autre côté, pour les hôpitaux, il est avantageux de remplir tous leurs postes vacants et ce, le plus tôt possible. Ceux-ci proposaient donc une place aux étudiants, lesquels ne sachant pas s'ils obtiendraient une meilleure offre, bien souvent acceptaient. Avait alors lieu une compétition entre les hôpitaux pour s'emparer des meilleurs éléments et remplir toutes les places vacantes. Les hôpitaux offraient un poste de plus en plus tôt aux étudiants de médecine et allaient jusqu'à proposer un poste aux étudiants de deuxième année. Les internes n'étaient pas toujours satisfaits de leur choix car leur poste ne correspondait pas à leurs attentes :

1. Consulter [GS62] et [RS90]

hôpital situé loin de leur domicile ou loin de leur conjoint, etc. Cela entraînait beaucoup de problèmes comme des ruptures de contrat. Le marché décentralisé ne fonctionnait pas du tout et c'est pourquoi une *clearing house* fut créée en 1952, qui depuis lors centralise toutes les demandes des hôpitaux ainsi que les listes des hôpitaux préférés des étudiants. Grâce à l'algorithme de NRMP [2] (*National Residence matching problem*), les étudiants sont alloués aux hôpitaux, de la manière la plus équitable possible afin que tout le monde y trouve son compte. D'ailleurs, en décembre 2012, Alvin Roth et Lloyd Shapley ont été récompensés par le prix Nobel d'économie pour leurs travaux théoriques et pratiques en théorie du *matching* et plus particulièrement pour leur contribution à trouver des solutions pratiques à un problème du monde réel.

Le principe est sensiblement le même qu'en *one-to-one matching*, si ce n'est qu'à présent on va assigner plusieurs agents à un seul. Dans le cas de l'allocation des élèves aux écoles, chaque école peut recevoir plusieurs élèves mais chaque élève ne peut être inscrit que dans une seule école. Une école sera acceptable pour un élève si celle-ci se trouve sur la liste de préférences de l'élève. Les écoles, quant à elles, doivent accepter tous les étudiants, dans les limites des places disponibles car ceux-ci ont le droit de s'inscrire dans n'importe quelle école. Cependant, la manière dont les écoles classent les élèves est intrinsèque et peut être une information privée. Les préférences sont toujours posées comme strictes et indépendantes : le fait qu'un élève préfère une école, n'influence pas les autres individus. Les écoles, quant à elles, ont des préférences séparables : elles préfèrent un élève à un autre et non pas un groupe d'élèves à un autre, même si cette hypothèse retire aux écoles une certaine homogénéité. En effet, si l'on remplace dans un groupe un élève par un individu qui lui est prioritaire, il n'est pas dit que le nouveau groupe ainsi formé travaillera mieux que l'ancien.

Le modèle du *many-to-one matching* consiste en

1. Un ensemble fini de n étudiants : $S = \{s_1, s_2, ..., s_n\}$
2. Un ensemble fini de m écoles : $C = \{c_1, c_2, ..., c_m\}$
3. Le nombre de places disponibles dans les écoles : $q = \{q_{c_1}, q_{c_2}, ..., q_{c_m}\}$
   avec q_j la capacité d'accueil de l'école c_j
4. Les préférences strictes des étudiants : $P_S = \{P_{s_1}, P_{s_2}, ..., P_{s_n}\}$
   $c_i P_{s_j}\ c_k$ signifie que l'étudiant s_j préfère l'école c_i à celle c_k
5. Les préférences strictes des écoles : $\succ_C = \{\succ_{c_1}, \succ_{c_2}, ..., \succ_{c_m}\}$
   $s_i \succ_{c_j}\ s_k$ signifie que l'école c_j préfère l'étudiant s_i à celui s_k

Le modèle du *many-to-one matching* se résume en

$$(S, C, q, P_S, \succ_C)$$

Le fait que les écoles possèdent des préférences séparables se traduit par la notion de réactivité restreinte.

2. Voir http://www.nrmp.org/

Définition 2.2.1. *Réactivité restreinte*

Pour $c \in C, \succ_c$ satisfait la réactivité restreinte (RR) si
$\forall S', S'' \subset S$ tels que $S' \succ_c \emptyset, S'' \succ_c \emptyset$ et $S'' = (S' - s') \cup s''$, avec $s' \in S', s'' \in S - S'$ ou $s'' = \emptyset$, nous avons $S' \succ_c S'' \Longleftrightarrow s' \succ_c s''$

A présent, redéfinissons un *matching* dans le cas du *many-to-one matching.*

Définition 2.2.2. *Matching*

Un matching μ est une fonction de l'ensemble $C \cup S$ dans l'ensemble $\mathcal{P}(C \cup S)$ tel que

- *$|\mu(s)| = 1, \forall s \in S$, tel que si $\mu(s) \neq s$, alors $\mu(s) \in C$*
- *$\mu(c) \subset S$ et $|\mu(c)| \leq q_c, \forall c \in C$*
  ($|\mu(c)|$ est le nombre d'élèves assignés pas le matching μ dans l'école c)
- *$\mu(s) = c \Longleftrightarrow s \in \mu(c)$*

Les deux définitions ci-dessus sont données par Abdulkadiroğlu dans [Abd05].

Remarque :

La fonction μ n'a pas un sens très mathématique car les ensembles, étudiants et écoles sont différents et que plusieurs étudiants peuvent être associés à une seule école. Il faudrait pour bien faire introduire une nouvelle fonction ou correspondance.
Soit la correspondance F,

$$\begin{array}{rll} F: & C \to\to & S \\ & c \rightsquigarrow & F(c) = \{s \in S : \mu(s) = c\} \end{array}$$

Dès lors, $F^{-1}(s) = c, \forall s \in F(c)$

Cependant, comme toute la littérature économique utilise la fonction μ, nous continuerons donc à l'utiliser par la suite.

Stabilité

Définition 2.2.3. *Matching stable*

Un matching μ est stable s'il n'existe pas de paire élève-école (s, c) telle que

- *$cP_s\mu(s)$ (L'élève s préfère l'école c à son école assignée $\mu(s)$) ;*
- *$\exists s', \mu(s') = c$ et $s \succ_c s'$ (Il y a un élève s' avec une priorité moindre que l'élève s, dans l'école c).*

Cette paire bloque le matching, elle est appelée blocking pair ou encore justified envy.

La stabilité peut être vue comme une forme de *justice* : un élève ne devrait pas être refusé dans une école de son choix s'il y est prioritaire par rapport aux autres étudiants inscrits.

Passons à l'utilité des *matching*. Pour pouvoir définir un *matching* S-optimal, introduisons la notation suivante :
Un élève s est indifférent entre deux *matching* (μ, μ') s'il obtient la même utilité dans les deux *matching*, donc la même école car les préférences sont strictes. Nous notons $\mu(s)R_s\mu'(s)$.

Définition 2.2.4. *Matching Pareto Optimal, Faiblement Pareto Optimal et S-Optimal*

Un matching μ est Pareto optimal s'il n'existe aucun matching μ' tel qu'au moins un élève préfère strictement μ' à μ et les autres sont indifférents entre les deux matching :

$$\nexists \mu' \text{ tel que } \exists s \in S : \mu'(s)P_s\mu(s) \text{ et } \mu(s')R_{s'}\mu'(s'), \forall s' \in S \setminus \{s\}$$

Un matching μ est faiblement Pareto optimal s'il n'existe aucun matching μ', tel que tous les élèves préfèrent strictement μ' à μ :

$$\nexists \mu' \text{ tel que } \forall s \in S : \mu'(s)P_s\mu(s)$$

On dira qu'un matching est optimal pour les étudiants (S-optimal) s'il est stable et Pareto optimal parmi les matching stables.

Les définitions ci-dessus sont tirées de [Pat11] et ne visent que l'utilité des étudiants.

Quant à la révélation des vraies préférences, ce sont Roth et Sotomayor qui étendirent le théorème de Gale et Shapley (Théorème 2.1.1) au cas du *many-to-one matching*. Ainsi, il n'existe pas de *matching* qui soit à la fois stable et *strategy-proof* pour tous les participants (écoles et étudiants). La preuve est similaire au Théorème 2.1.1, il n'est donc pas nécessaire de la présenter ici mais elle se trouve dans [Abd05].

Trois algorithmes sont principalement utilisés pour assigner les élèves aux écoles.

Student-Applying Deferred Acceptance Algorithm

Le premier est le *Student-Applying Deferred Acceptance Algorithm* (SDAA) dérivé du *deferred acceptance algorithm* en *one-to-one matching*. Le NMRP (*National Resident Matching Program*) en est un cas particulier.

1. Etape 1, chaque étudiant s'inscrit dans l'école qu'il préfère. Les écoles assignent provisoirement une place aux élèves selon leur ordre de priorité et rejettent les élèves qui dépassent le nombre de places disponibles. Les élèves acceptés sont sous réserve ;
2. Etape $k \geq 2$, chaque élève qui a été rejeté à l'étape $k-1$ postule dans l'école suivante sur sa liste de préférences. Chaque école attribue une place provisoire aux élèves, en choisissant parmi les nouveaux postulants et ceux en réserve et ce, sans dépasser les places disponibles. Elle rejette ensuite les élèves en trop ;
3. L'algorithme s'arrête lorsque tous les élèves sont inscrits dans une école ou qu'ils ont épuisé leur liste d'écoles admissibles.

Remarque :

Dans la réalité, aucun élève ne finira sans école, nous en parlerons lorsque nous analyserons le décret des inscriptions dans la Section 5.1.

On peut prouver que cet algorithme donne un *matching* stable, la preuve est analogue à celle pour le DAA (Théorème 2.1.1) et nous laissons au lecteur le soin de le vérifier. De plus, il n'existe aucun autre *matching* stable qui domine le *matching* trouvé par le SDAA. Ce *matching* est l'unique *matching* S-optimal.

Nous présentons ici un résultat important démontré par Alvin Roth dans [Rot85].

Théorème 2.2.1.

Si un algorithme donne toujours le matching S-optimal, quelles que soient les préférences des élèves, alors révéler leurs vraies préférences est la stratégie dominante pour ces derniers.

Il n'existe aucun mécanisme qui soit strategy-proof pour les écoles.

Le SDAA est dès lors strategy-proof pour les étudiants mais pas pour les écoles.

Top Trading Cycles

Définition 2.2.5. *Cycle*

En théorie des graphes, un cycle est une suite d'arêtes consécutives dont les sommets extrémités sont identiques.

Le deuxième algorithme étudié est l'algorithme de *Top Trading Cycles* (TTC).

1. Etape 1, chaque élève nomme l'école qu'il préfère et chaque école désigne l'élève le plus haut placé dans sa liste. Un graphe est formé de la sorte : les noeuds sont les élèves et les écoles, une arrête relie deux noeuds lorsqu'un élève a choisi une école et vice-versa. Il y a au moins un cycle formé par des élèves et des écoles et chaque élève fait partie d'au plus un cycle (les cycles formés sont disjoints). Pour chaque cycle, on attribue chaque élève à l'école qu'il a choisie. Les quotas des écoles sont diminués de 1 et si le quota atteint 0, l'école est complète et sort du jeu.
2. Etape $k \geq 2$, le processus recommence, les élèves désignent leur école préférée et les écoles font de même (tant qu'il y a de la place dans une école, elle reste sur la liste des étudiants, sinon elle en est effacée). De même, il existe au moins un cycle et chaque élève dans le cycle est assigné à l'école vers laquelle il pointe.
3. L'algorithme se termine lorsque tous les élèves sont assignés à une école ou qu'ils ont épuisé leur liste d'écoles.

Ce mécanisme est strategy-proof pour les étudiants car il est direct et il produit également un *matching* qui est Pareto optimal [3] mais pas stable.

3. Pour plus de détails, consulter [AS03]

Algorithme de Boston

Le troisième algorithme doit son nom à la ville de Boston, où il a été utilisé pour attribuer les étudiants aux universités, jusqu'en 2005. Le mécanisme est le suivant :

1. Etape 1, on assigne à chaque étudiant son premier choix, dans les limites du possible, c'est-à-dire s'il reste de la place dans l'école et suivant l'ordre de priorité de celle-ci.
2. Etape $k \geq 2$, les élèves rejetés considèrent leur choix suivant dans leur liste d'écoles et celles-ci les acceptent selon les mêmes critères que précédemment.

Malheureusement, il est facile de voir que cet algorithme n'incite pas les étudiants à donner leurs vraies préférences. En effet, si une école est très prisée, on peut supposer que seuls les meilleurs élèves seront admis ; si un participant ne fait pas partie de ceux-ci, il est inutile qu'il mette cette école comme premier choix. Car si au premier tour, il n'est pas admis dans cette école, il se pourrait que l'école de son second choix soit également complète au second tour et le refuse. Mettre une école très prisée comme premier choix, est un risque dangereux à prendre et si l'étudiant est prêt à le prendre, il devrait jouer la carte de la prudence et mettre comme second choix, une école accessible. Dès lors, cet algorithme ne donne pas un *matching strategy-proof*. Cependant, par construction, ce *matching* est Pareto-optimal, même s'il n'est pas stable.

Pour mieux comprendre comment fonctionnent ces algorithmes, voici un exemple avec 3 écoles et 4 élèves qu'il faut assigner aux écoles.

Exemple 2.2.1.

1. $C = \{c_1, c_2, c_3\}$
2. $S = \{s_1, s_2, s_3, s_4\}$
3. Chaque école peut accepter un élève, sauf la troisième école qui peut en accepter 2, $q = \{1, 1, 2\}$

4. Supposons que les préférences des élèves soient données par :

P_{s_1}	P_{s_2}	P_{s_3}	P_{s_4}
c_2	c_1	c_1	c_2
c_1	c_2	c_2	c_3
c_3	c_3	c_3	c_1

TABLE 2.6 – Exemple 2.2.1, préférences des élèves

5. Supposons que les préférences des écoles soient données par :

P_{c_1}	P_{c_2}	P_{c_3}
s_1	s_3	s_3
s_3	s_4	s_1
s_2	s_2	s_4
s_4	s_1	s_2

TABLE 2.7 – Exemple 2.2.1, préférences des écoles

A On applique le *Student-Applying Deferred Acceptance Algorithm.*

Première étape, les élèves proposent à leur premier choix :

$$\begin{array}{cccc} \mu_1 & c_1 & c_2 & c_3 \\ & s_3 & s_4 & \end{array}$$

Deuxième étape, les élèves s_1 et s_2 proposent à leur second choix :

$$\begin{array}{cccc} \mu_2 & c_1 & c_2 & c_3 \\ & s_1 & s_4 & \end{array}$$

Troisième étape, s_2 propose à son dernier choix et s_3 à son deuxième :

$$\begin{array}{cccc} \mu_3 & c_1 & c_2 & c_3 \\ & s_1 & s_3 & s_2 \end{array}$$

Quatrième étape, s_4 propose à son deuxième choix :

$$\begin{array}{cccc} \mu_3 & c_1 & c_2 & c_3 \\ & s_1 & s_3 & s_2 \\ & & & s_4 \end{array}$$

Ce matching est stable et strategy-proof

B On applique le Top Trading Cycles.

Première étape, on construit les cycles :

$$\begin{array}{ccccccc} s_1 & \longrightarrow & c_2 & \longleftarrow & s_4 & & c_3 \\ \uparrow & & \downarrow & & & & \\ c_1 & \longleftarrow & s_3 & & & & \\ \uparrow & & & & & & \\ s_2 & & & & & & \end{array}$$

On remarque un cycle entre s_1, c_2, s_3, c_1 et on attribue à chaque élève dans ce cycle, son école, ce qui nous donne :

$$\begin{array}{cccc} \mu_1 & c_1 & c_2 & c_3 \\ & s_3 & s_1 & \end{array}$$

Comme il n'y plus de places disponibles dans les écoles 1 et 2, les élèves s_2 et s_4 sont assignés à l'école 3.

μ_2	c_1	c_2	c_3
	s_3	s_1	s_2
			s_4

Ce matching n'est pas stable, en effet, la paire (s_4, c_2) bloque le matching car s_4 préférerait l'école c_2 à c_3 et il est prioritaire sur s_1 dans l'école c_2.

C On applique l'algorithme de Boston

s_1 et s_4 postulent pour l'école 2, or celle-ci préfère s_4, donc on attribue l'école c_2 à s_4. Egalement, s_2 et s_3 postulent pour l'institution c_1 qui préfère s_3.

μ_1	c_1	c_2	c_3
	s_3	s_4	

Les élèves restants sont alloués à l'école c_3 puisqu'il n'y a plus de place nulle part ailleurs.

μ_1	c_1	c_2	c_3
	s_3	s_4	s_1
			s_2

On voit très facilement que ce *matching* n'est pas stable, car il existe une blocking pair (s_1, c_1). En effet, s_1 préfère l'école c_1 à l'école c_3 et il est prioritaire sur s_3 dans l'école c_1.

Dans cet exemple, nous appliquons les 3 algorithmes différents et trouvons 3 *matching*, qui pourtant sont tous 3 optimaux, car il existe plusieurs *matching* Pareto-optimaux. Cependant, il n'existe qu'un matching S-optimal (pour un profil de préférences donné), qui est Pareto-optimal et *matching* stable.

En résumé, les 3 algorithmes les plus utilisés en théorie du *many-to-one matching* sont le *(Student-Applying) Deferred Acceptance Algorithm* (SDAA), le *Top Trading Cycles* (TTC) et l'algorithme de Boston.

Propriétés / Algorithme	Stabilité	DSIC [4]	Efficience
SDAA	oui	oui pour les étudiants	Unique S-optimal
TTC	non	oui pour les étudiants	Pareto optimal
Boston	non	non	Pareto optimal

TABLE 2.8 – Propriétés des algorithmes

Un *matching* ne peut pas être à la fois Pareto optimal, stable et *strategy-proof* (résultat tiré de [AS03]) ce qui implique qu'un *matching* s'il est *strategy-proof* et stable perd immanquablement de l'efficience. Le fait de révéler ses vraies préférences apporte en général une

4. DSCI = *Dominant Strategy Incentive Compatible*, autre nom donné à un *matching strategy-proof*

perte d'efficience, comme par exemple, des places vides dans certaines écoles etc. Il faut donc choisir si l'on préfère l'efficience ou la stabilité.

Dans ce système, les écoles n'ont pas grand chose à dire par rapport à l'allocation des élèves dans leurs locaux mais elles peuvent tout de même les classer selon leurs préférences. Celles-ci peuvent être non homogènes et engendrer des discriminations entre les élèves. De plus, comme chaque école adopte sa propre politique, les critères de sélection des élèves peuvent varier beaucoup. Ainsi, de multiples problèmes apparaissent, comme la ségrégation sociale, aussi bien sur le plan financier que sur le plan racial. C'est pourquoi, bien souvent, l'Etat ou la ville impose des quotas afin d'obtenir dans chaque école un équilibre entre les élèves, qu'il soit racial, ethnique ou socio-économique.

2.3 Affirmative action

Les exigences de l'Etat visant à harmoniser les écoles se traduisent par des quotas ajoutés au problème du *many-to-one matching*. Ces contraintes sont de type ethnique, socio-économique ou portent sur le genre (masculin ou féminin) ou le district (endroit où les élèves habitent). Elles peuvent même être une combinaison de tous ces critères. Avant tout, il faut attribuer un "type" à chaque étudiant afin de pouvoir classer ceux-ci par après. Ensuite, les écoles se verront attribuer des quotas pour chaque type d'élèves.

Après introduction des types et des quotas, le problème devient donc :

1. Un ensemble fini de n étudiants : $S = \{s_1, s_2, ..., s_n\}$
2. Un ensemble fini de m écoles : $C = \{c_1, c_2, ..., c_m\}$
3. Le nombre de places disponibles dans les écoles : $q = \{q_{c_1}, q_{c_2}, ..., q_{c_m}\}$
   Avec q_j la capacité d'accueil de l'école j
4. Les préférences strictes des étudiants : $P_S = \{P_{s_1}, P_{s_2}, ..., P_{s_n}\}$
   $c_i P_{s_j}\ c_k$ signifie que l'étudiant s_j préfère l'école c_i à celle c_k
5. Un ensemble de k types : $T = \{\tau_1, \tau_2, ..., \tau_k\}$
6. Une fonction type : $f : S \to T$, $f(s) = \tau_j$, τ_j est le type de s
7. Les préférences strictes des écoles : $\succ_C = \{\succ_{c_1}, \succ_{c_2}, ..., \succ_{c_m}\}$
   $s_i \succ_{c_j}\ s_k$ signifie que l'école c_j préfère l'étudiant s_i à celui s_k
8. Pour chaque école $c \in C$, un vecteur de quotas maximaux pour chaque type :
   $q_c^T = (q_c^{\tau_1}, q_c^{\tau_2}, ..., q_c^{\tau_2})$
   Avec $q_c^{\tau_i} \leq q_c, \forall i \in \{1, ..., k\}$
   $Q = (q_{c_i}^T), i \in \{1, ..., m\}$

Un problème de *school choice* avec affirmative action se résume en

$$(S, C, q, T, f, P_S, \succ_C, Q)$$

Les quotas imposés sur les types feront désormais partie des préférences des écoles. Cependant, il faut bien avoir en tête que si ces quotas sont imposés pour un meilleur fonctionnement des établissements scolaires, une déségrégation sociale ou encore pour d'autres raisons, il n'empêche qu'ils violent les préférences des étudiants, et par là diminuent l'efficience de n'importe quel

algorithme, mais nous en reparlerons plus en détail dans la Section 2.5.
Du fait de cette perte d'efficience, quelques notions doivent être révisées.

Justice / stabilité

Définition 2.3.1. *Affirmative Action (AA)*

Une liste d'élève $S' \subset S$ respecte les contraintes d'affirmative action pour l'école $c \in C$ si $|\{s \in S' : f(s) = \tau_j\}| \leqslant q_c^{\tau_j},\ \forall j \in \{1..k\}$

Une école $c \in C$ respecte les contraintes d'affirmative action si
$\forall S', S'' \subset S$, *tel que S' respecte les contraintes d'affirmative action pour c et S'' ne les respecte pas, alors $S' \succ_c S''$*

Définition 2.3.2. *Justice ou stabilité*[5]

Un matching est stable si

1. $\forall c \in C, |\mu(c)| \leq q_c$, $\mu(c)$ *respecte* q_c^T *et* $\forall s \in S, \mu(s)$ *est acceptable pour* s
2. *Il n'existe pas d'étudiant $s \in S$ ni d'école $c \in C$ tel que $cP_s\mu(s)$ et*
   - $\mu(c) \cup s$ *respecte la capacité de c et les contraintes de c ;*
     ou bien
   - $\exists s' \in S : \mu(s') = c, (\mu(c) - s') \cup s$ *respecte les contraintes de c et $s \succ_c s'$.*

Un *matching* stable respecte les préférences des écoles, dans les limites des contraintes.

Les deux définitions ci-dessus sont proposées par Abdulkadiroğlu dans [Abd05].

A présent examinons les propriétés de ce genre de problème.

Proposition 2.3.1.

Si les préférences des écoles satisfont la réactivité restreinte (Définition 2.2.1) et les contraintes d'affirmative action (Définition 2.3.1), alors l'ensemble des matching stables (Définition 2.3.2) est non vide.

De plus, le Student-Appying Deferred Acceptance Algorithm produit un matching stable, que les étudiants aiment au moins autant que n'importe quel autre matching stable (S-optimal).

On remarque que avec l'apparition des contraintes, la notion de stabilité est moins forte mais néanmoins, le *matching* produit par SDAA est S-optimal et nous aurons l'existence d'un *matching* stable et *strategy-proof* sous certaines conditions. Le théorème suivant est présenté par Abdulkadiroğlu, dans [Abd05]

5. Dans la littérature, on rencontre plus souvent la notion de justice alors qu'au final, ce n'est qu'une extension de la stabilité du cas sans contraintes et pour plus de facilité, nous ne parlerons que de stabilité par la suite.

Théorème 2.3.1.

Lorsque les préférences des écoles satisfont la réactivité restreinte (Définition 2.2.1) et les contraintes d'affirmative action (Définition 2.3.1), l'algorithme SDAA incite les participants à révéler leurs vraies préférences ; autrement dit, il produit un matching strategy-proof.

Pour démontrer le Théorème 2.3.1, nous aurons besoin de 2 lemmes, le lecteur intéressé trouvera leur preuve dans [Abd05].

Introduisons les notations suivantes :

Notations :

Soit $s \in S$, P_s sont les préférences réelles de s et P_{-s} les préférences réelles des autres joueurs, $S \setminus \{s\}$. Après l'application du *Student-Appying Deferred Acceptance Algorithm*, on obtient un *matching* μ, noté $DA^S(P_{-s}, P_s)$.
Soit Q_s, les préférences stratégiquement modifiées de s et ν le matching obtenu après application du *Student-Appying Deferred Acceptance Algorithm* avec les nouvelles préférences de s, noté $DA^S(P_{-s}, Q_s)$ (P_{-s} reste inchangé).

Lemme 2.3.1.

Soit Q_s, si $\nu(s)P_s\mu(s)$, alors $\forall s' \in S, \nu(s')P_{s'}\mu(s')$

Lemme 2.3.2.

Soit Q_s, si $\nu(s)P_s\mu(s)$, alors $|\nu(c)| = |\mu(c)|, \forall c \in C$

A présent, démontrons le **Théorème 2.3.1**

Démonstration.

Soit
- μ, le *matching* obtenu avec les préférences réelles ($DA^S(P_{-s}, P_s)$), obtenu après t étapes
- $s \in S$ qui modifie ses préférences en Q_s, les autres élèves gardent leurs préférences identiques
- ν le *matching* obtenu avec les nouvelles préférences ($DA^S(P_{-s}, Q_s)$)

On aura soit $\nu(s)P_s\mu(s)$ ou bien $\nu(s) = \mu(s)$.

Montrons que $\nu(s)P_s\mu(s)$ est impossible.

Soit $s' \in S$ admis dans l'école c à l'étape t.
Montrons que $\nu(s') = \mu(s')$.

On observe que

I Aucun étudiant n'est rejeté par c en t (sinon, t ne serait pas la dernière étape de l'algorithme)

II Il y a toujours au moins une place libre dans c, avant l'étape t (par I)

III Le quota limite dans c pour le type de s' ($f(s')$) n'est jamais atteint avant l'étape t (sinon s' ne serait pas admis en t, par I et II)

IV $\forall \hat{s} \in (S \setminus s') : f(\hat{s}) = f(s')$ n'est jamais rejeté par c avant l'étape t (par III)

V Si un élève $\hat{s} \in (S \setminus s')$ est rejeté à l'étape $k < t$, alors $f(\hat{s}) \neq f(s')$ et le quota limite de $f(\hat{s})$ est atteint en t (par IV)

Supposons par l'absurde que $c \neq \nu(s')$

Par le **Lemme 2.3.1**, comme $\nu(s) P_s \mu(s)$, alors $\nu(s') P_{s'} c$, ce qui suppose que s' n'a pas proposé à c dans $DA^S(Q_s)$

Par le **Lemme 2.3.2**, $|\nu(c)| = |\mu(c)|$.
Donc, $\exists s'' \in (S \setminus s') : \mu(s'') \neq c \wedge \nu(s'') = c \wedge c P_{s''} \mu(s'')$.

Quel est le type de s'' ?

Si $f(s'') = f(s')$ et $c P_{s''} \mu(s'')$, s'' aurait proposé à c avant de proposer à $\mu(s'')$, dans $DA^S(P)$ et aurait été accepté (par IV), ce qui contredit $\mu(s'') \neq c$.

Donc $f(s'') \neq f(s')$ et si $c P_{s''} \mu(s'')$ alors par V, s'' a été rejeté par c dans $DA^S(P)$.

Dans $\nu = DA^S(Q_s)$, s'' a pris la place de $\hat{s}$ dans c, tel que $f(\hat{s}) = f(s'') \neq f(s')$ et $c = \mu(\hat{s}) \neq \nu(\hat{s})$.
Donc la place qui était libre à l'étape $t-1$ pour le type $f(s')$ dans $DA^S(P)$ n'est pas complétée dans $DA^S(Q_S)$, ce qui implique que $|\nu(c)| < |\mu(c)|$ et contredit le **Lemme 2.3.2**.

On vient de prouver que $\forall s'$ admis dans une école en t, $\mu(s') = \nu(s')$.

Le reste de la preuve se fait par induction.

L'hypothèse d'induction est que $\forall k < t, \mu(s') = \nu(s'), \forall s'$ admis dans une école en $k+1$ ou plus tard.

On vient juste de le montrer pour $k = t - 1$, à présent pour k.

Supposons par l'absurde que s' est admis dans c, à l'étape k sous $DA^S(P)$ et que $c = \mu(s') \neq \nu(s')$.

Par le **Lemme 2.3.1**, $\nu(s') P_{s'} c$ donc s' ne propose pas à c dans $DA^S(Q_s)$.
Soit $S'' = \{s'' : \nu(s'') = c \neq \mu(s'')\}$.

Par le **Lemme 2.3.2**, $|\nu(c)| = |\mu(c)|$, donc $S'' \neq \emptyset$.

Par le **Lemme 2.3.1**, $c P_{s''} \mu(s''), \forall s'' \in S''$.
Donc $\exists \hat{s} \in S''$ qui est rejeté par c en faveur de s' dans $DA^S(P)$. Soit $\hat{s}$ était accepté par c et est rejeté à l'étape k en faveur de s' et dans ce cas, il est admis dans une école à l'étape

$k' > k$, soit $\hat{s}$ fait sa demande à l'étape $k' > k$ à c mais est rejeté et sera admis dans une école à l'étape $k'' > k' > k$. Dans les deux cas, $\hat{s}$ ne sera admis dans une école qu'après l'étape k de $DA^S(P)$ et on peut appliquer l'hypothèse d'induction, $\mu(\hat{s}) = \nu(\hat{s})$ ce qui contredit le fait que $\hat{s} \in S''$.

On a bien prouvé que $\forall s' \in S, \mu(s') = \nu(s')$, et en particulier pour s.

Dès lors, s n'a aucun avantage à manipuler ses préférences et l'algorithme *Student-Appying Deferred Acceptance Algorithm* donne un *matching strategy-proof*.

□

Si le but est d'équilibrer les écoles, il n'est pas toujours atteint avec ce genre de contraintes. En effet, prenons l'exemple de la déségrégation sociale en Amérique. Une école peut accueillir 100 élèves et l'on impose des quotas de 50 noirs et 50 blancs maximum. Alors si cette école est composée de seulement 50 noirs, elle respecte ses quotas mais cela ne veut pas dire qu'elle est équilibrée et l'objectif visé n'est pas obtenu. Le problème est qu'on ne dispose d'aucune information à propos du nombre minimum d'élèves de chaque type. Pour le résoudre, il faut donc introduire des quotas minimaux, c'est ce que nous ferons dans la section suivante : '*Controlled school choice*'.

2.4 *Controlled school choice*

Si nous introduisons des règles à propos d'un nombre minimal de chaque type d'élèves, nos quotas deviennent des intervalles.

On peut modéliser le nouveau cas comme suit :

1. Un ensemble fini de n étudiants : $S = \{s_1, s_2, ..., s_n\}$
2. Un ensemble fini de m écoles : $C = \{c_1, c_2, ..., c_m\}$
3. Le nombre de places disponibles dans les écoles : $q = \{q_{c_1}, q_{c_2}, ..., q_{c_m}\}$
   Avec q_j la capacité d'accueil de l'école j
4. Les préférences strictes des étudiants : $P_S = \{P_{s_1}, P_{s_2}, ..., P_{s_n}\}$
   $c_i P_{s_j}\ c_k$ signifie que l'étudiant s_j préfère l'école c_i à celle c_k
5. Un ensemble de k types : $T = \{\tau_1, \tau_2, ..., \tau_k\}$
6. Une fonction type : $f : S \to T$, $f(s) = \tau_j$, τ_j est le type de s
7. Les préférences strictes des écoles : $\succ_C = \{\succ_{c_1}, \succ_{c_2}, ..., \succ_{c_m}\}$
   $s_i \succ_{c_j}\ s_k$ signifie que l'école c_j préfère l'étudiant s_i à celui s_k
8. Pour chaque école, deux vecteurs de quotas pour chaque type : $\underline{q}_c^T = (\underline{q}_c^{\tau_1}, \underline{q}_c^{\tau_2}, ..., \underline{q}_c^{\tau_2})$ la limite inférieure des quotas et $\overline{q}_c^T = (\overline{q}_c^{\tau_1}, \overline{q}_c^{\tau_2}, ..., \overline{q}_c^{\tau_2})$ la limite supérieure des quotas
   Avec $\underline{q}_c^{\tau_i} \leq \overline{q}_c^{\tau_i} \leq q_c, \forall i \in \{1, ..., k\}$
   $Q = (\underline{q}_{c_i}^T, \overline{q}_{c_i}^T), i \in \{1, ..., m\}$

Un problème de *controlled school choice* se résume en

$$(S, C, q, T, f, P_S, \succ_C, Q)$$

Les quotas, comme vu précédemment, peuvent être simples, comme le sexe, composé de {masculin, féminin}, ou encore l'origine ethnique, par exemple composée de {blanc, noir, asiatique}, mais ils peuvent être une combinaison de ces critères, comme {masculin, féminin} × {blanc, noir, asiatique}. Nous ne le démontrerons pas, mais tous les résultats qui suivent s'appliquent également dans le cas des quotas multi-dimensionels [6].

Définition 2.4.1.

Un ensemble d'élève $S' \subset S$ respecte les contraintes (de controlled choice) pour l'école $c \in C$ si $\underline{q}_c^{\tau_j} \leq |\{s \in S' : f(s) = \tau_j\}| \leq \overline{q}_c^{\tau_j}$, $\forall j \in \{1..k\}$

Un matching μ respecte les contraintes, pour chaque école $c \in C$ si $\underline{q}_c^{\tau_j} \leq |\mu^{\tau_j}(c)| \leq \overline{q}_c^{\tau_j}$, $\forall j \in \{1..k\}$, avec $\mu^{\tau_j}(c) = \{s \in \mu(c) : f(s) = \tau_j\}$

Remarques :

1. Si les quotas sont donnés en pourcentages, il suffit de multiplier ceux-ci par le nombre d'élèves assignés dans chaque école.

2. Afin de ne pas se retrouver dans des cas d'impossibilité, nous ferons l'hypothèse que le nombre de participants de chaque type est plus élevé que la somme des quotas minimum de ce type, dans toutes les écoles

$$|S^{\tau_j}| \geq \sum_{c \in C} \underline{q}_c^{\tau_j}, \qquad \forall j \in \{1..k\}$$

   Avec, $S^{\tau_j} = \{s \in S : f(s) = \tau_j\}$

3. On supposera également qu'il y a une place pour chaque élève dans les écoles, autrement dit, que le nombre d'élèves de chaque type ne dépasse pas la somme des quotas maximaux de ce type, dans toutes les écoles

$$|S^{\tau_j}| \leq \sum_{c \in C} \overline{q}_c^{\tau_j}, \qquad \forall j \in \{1..k\}$$

Les contraintes peuvent être vues de deux manières différentes. La première consiste à respecter scrupuleusement les quotas et assigner les élèves seulement en se basant sur ce critère, cette méthode s'appelle *Hard Bounds*. Un second point de vue, quant à lui plus laxiste, existe et considère les quotas comme des lignes de conduites plutôt que comme des règles fixes. Cette approche est appelée *Soft Bounds*.

Hard Bounds

Le fait que les règles doivent être respectées rigoureusement signifie que les quotas minimaux doivent être dépassés et que les quotas maximaux ne peuvent être outrepassés. De

6. Pour plus d'information, consulter [Ehl10]

plus, il est logique de demander que les *matching* respectent la définition suivante qui établit qu'une place vide ne devrait pas le rester s'il existe un élève en droit de la réclamer.

Définition 2.4.2. *Non-wastefulness*

On dira qu'un élève s est en droit de réclamer une place vide dans l'école c, en supposant qu'on est dans le matching μ, si

1. $cP_s\mu(s)$ *et* $|\mu(c)| \leq q_c$;
2. $|\underline{q}^{f(s)}_{\mu(s)}| < |\mu^{f(s)}(\mu(s))|$ *(dans $\mu(s)$, le quota minimal du type de s est strictement dépassé) ;*
3. $|\mu^{f(s)}(c))| < |\overline{q}^{f(s)}_{c)}|$ *(dans c, le quota maximal du type de s n'est pas atteint).*

On dira qu'un matching est non-wasteful si aucun élève n'est en droit de réclamer une place dans aucune école.

La stabilité doit maintenant tenir compte des contraintes : pour qu'un étudiant puisse en remplacer un autre, non seulement le premier doit avoir une priorité plus élevée mais également le nouveau *matching* ne peut pas violer les contraintes.

Définition 2.4.3. *Stabilité*

Un matching est stable si
$\nexists s \in S, \nexists c \in C$ *et* $, s' \in \mu(c)$ *tel que* $cP_s\mu(s)$ *et* $s \succ_c s'$ *et*

$(\mu(c) - s') \cup s$ *respecte les contraintes de c et s' peut être assigné à une école, sans que cela ne viole les contraintes.*

Nous énonçons sans démontrer un premier résultat négatif confirmant la perte de justice à laquelle on s'attendait en ajoutant les quotas minimaux.

Théorème 2.4.1. *L'ensemble des matching stables peut être vide.*

Dès lors, on décide de reconsidérer la notion de stabilité et de seulement considérer un *matching* stable entre les différents types.

Définition 2.4.4. *Stabilité entre les types*

Un étudiant s peut remplacer un élève s' dans l'école c, en supposant la matching μ, si

1. $\mu(s') = c, \quad cP_s\mu(s)$ *et* $s \succ_c s'$;
2. $f(s) = f(s')$.

Un matching sera stable entre les types si aucun élève ne peut en remplacer un autre du même type.

Les définitions ci-dessus sont données par L. Ehlers dans [Ehl10].

Malheureusement, nous n'avons encore aucun résultat satisfaisant.

Théorème 2.4.2.

L'ensemble des matching qui sont à la fois stables entre les types et non-wasteful peut être vide.

Ce résultat négatif est tiré de [EHYY12], et afin d'obtenir de meilleures conclusions, il faut de nouveau introduire une nouvelle notion. Comme la stabilité entre les types est déjà assez faible et qu'elle est plus importante que la propriété de *non-wastefulness*, c'est cette dernière que nous atténuons.

Définition 2.4.5. *Constrained non-wasteful*

Un matching μ constrained non-wasteful est tel que :
Si un élève s est en droit de réclamer une place vide dans l'école c, en supposant le matching μ,
alors le matching μ' n'est pas stable entre les types, avec μ' tq $\mu'(s) = c$ et $\mu'(s') = \mu(s'), \forall s' \in S \setminus \{s\}$

A présent, le résultat suivant est possible :

Théorème 2.4.3.

L'ensemble des matching étant à la fois stables entre les types (Définition 2.4.4) et constrained non-wasteful (Définition 2.4.5) est non vide.

La démonstration se trouve dans [Ehl10].

Si l'on modifie l'algorithme de *deferred acceptance*, on obtient un algorithme qui produit un *matching* stable et *constrained non-wasteful.*

Controlled Student Proposing Deferred Acceptance Algorithm

Avant de commencer, il faut assigner aléatoirement un numéro à chaque étudiant, disons : $\{s_1, s_2, ..s_n\}$.
Soit $\mathcal{F}$ l'ensemble des *matching* stables entre les types et *constrained non-wasteful.*
L'algorithme consiste à choisir parmi les *matching* de $\mathcal{F}$, celui qui va satisfaire le plus de participants possible, cela se fera par étape et à chaque pas, il y aura un *matching* provisoire ν_k.
On pose $\nu_0 \in \mathcal{F}$, la première allocation, telle que $\forall s \in S, \nu_0(s) = s$.

1. A l'étape 1, l'élève s_1 propose à son école préférée, disons c_1.

   i Si $\exists \mu \in \mathcal{F}$, tq $\mu(s_1) = c_1$ alors, $\nu_1(s_1) = c_1 \wedge \nu_1(s) = \nu_0(s) = s, \forall s \in S \setminus \{s_1\}$

   ii Sinon, s_1 est rejeté par l'école c_1 et $\nu_1 = \nu_0$

2. A l'étape $k \geq 2$, si $\exists s$ tq $\nu_{k-1}(s) = s$, et supposons que l'élève s est celui dont l'index est minimum, parmi les participants non assignés à une école. Soit c, son école préférée (parmi celles auxquelles il n'a pas encore proposé).

   i Si $\exists \mu \in \mathcal{F}$ tq $\mu(s) = c \wedge \mu(s') = \nu_{k-1}(s'), \forall s'$ tq $\nu_{k-1}(s') \neq s'$ alors, l'étudiant s est en droit de réclamer une place dans l'école c et $\nu_k(s) = c \wedge \nu_k(s') = \nu_{k-1}(s'), \forall s' \in S \backslash \{s\}$

   ii Si 2i est faux, mais si $\exists s'$ tq $f(s) = f(s')$ (sous le *matching* ν_{k-1}) et tel que s est prioritaire sur s' dans c ($s \succ_c s'$), alors $\nu_k(s) = c \wedge \nu_k(s') = s' \wedge \nu_k(s'') = \nu_{k-1}(s''), \forall s'' \in S \setminus \{s, s'\}$. L'étudiant s' est donc rejeté et devra retenter sa chance à la prochaine étape.

   iii Si ni 2i, ni 2ii ne sont vraies, alors $\nu_k = \nu_{k-1}$ et l'élève s est rejeté par c.

L'algorithme se termine à l'étape t, quand $\nu_t(s) \neq s, \forall s \in S$ et $\nu_t = \mu$, le *matching* final pour l'algorithme *Controlled Student Proposing Deferred Acceptance Algorithm* (CDAA).

Que peut-on dire à présent sur le *matching* obtenu ? Est-il stable, quelle efficience apporte-t-il ?

Théorème 2.4.4.

Pour n'importe quel problème de controlled school choice, le CDAA produit un matching qui est stable entre les types et constrained non-wasteful.

Pour n'importe quel problème de controlled school choice, le CDAA produit un matching qui est faiblement Pareto-optimal.

La preuve de ce théorème se trouve dans [Ehl10] mais nous ne la voyons pas ici.

Remarque :

Le fait que le *matching* soit faiblement Pareto-optimal signifie qu'il n'existe aucun *matching* qui soit strictement préféré par tous les étudiants à ce *matching*. Cela ne veut pas dire qu'il est S-optimal car certains étudiants peuvent obtenir une meilleure place, en échangeant d'école (entre eux), alors que les autres élèves restent dans la même école (leur bien-être ne diminue pas). C'est d'ailleurs l'objet de l'algorithme suivant.

Controlled Student Proposing Deferred Acceptance Algorithm with Improvement (CDAAI)

On peut ajouter une nouvelle phase à cet algorithme. Celle-ci recherche l'amélioration de l'efficience, parmi les *matching* stables entre les types.

La première étape est la même que dans le *Controlled Student Proposing Deferred Acceptance Algorithm*.

La seconde étape est la suivante :

1. Construction d'un graphe pour le *matching* μ. Pour chaque école, on crée k noeuds, un pour chaque type, comme $c(\tau_1), ..., c(\tau_k)$. S'il reste des places libres dans une école, on crée un noeud supplémentaire représentant celles-ci, $c(\tau_0)$.

2. Pour chaque étudiant de type τ_i et école c, on considère les étudiants du même type qui préfèreraient être assignés à c plutôt qu'à leur école actuelle. C'est-à-dire, $S' = \{s \in S^{\tau_i} : cP_s\mu(s)\}$. Si cet ensemble est vide, il n'y a rien à faire. Sinon, on considère l'étudiant dans cet ensemble qui est prioritaire selon c. Soit $s \in S'$, alors $\mu(s)(\tau_i)$ pointe vers $c(\tau_i)$. En plus, $\mu(s)(\tau_i)$ pointe aussi vers $c(\tau_j)$ tel que s peut être admis dans c en remplaçant un élève de type τ_j (seulement si $|\mu^{\tau_j}(c)| > \underline{q}_c^{\tau_j}$ et $|\mu^{\tau_i}(c)| < \overline{q}_c^{\tau_i}$). Si il y a une place libre dans c, alors $\mu(s)(\tau_i)$ pointe également vers $c(\tau_0)$, si s peut prendre cette place, sans violer les quotas de c.
3. Pour chaque école c possédant des places libres, $c(\tau_0)$ pointe vers chaque $c'(\tau_i)$, où $c' \neq c$ et $\tau_i \neq \tau_0$ et tel que un étudiant de type τ_i peut être enlevé de c', sans violer les contraintes de c' $(\mu^{\tau_i}(c')| > \underline{q}_c'^{\tau_i})$.
4. S'il n'existe pas de cycle dans le graphe, alors on arrête. Sinon, on réassigne les étudiants associés avec chaque noeud du cycle. Les nouvelles correspondances deviennent le *matching* μ et on recommence à l'étape 1.

Cependant, avec l'amélioration, il se pourrait que le CDAA trouve des *matching* différents en fonction de l'ordre assigné aux étudiants. Mais grâce à cette phase, on obtient un *matching* S-optimal.

Théorème 2.4.5.

Pour n'importe quel problème de controlled school choice, le CDAAI produit un matching qui est stable entre les types et constrained non-wasteful. De plus, ce matching est S-optimal parmi les matching stables entre les types et constrained non-wasteful.

Ce théorème est démontré dans [EHYY12].

Pour mieux comprendre le fonctionnement du CDAAI, voici un exemple.

Exemple 2.4.1.

1. $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$
2. $C = \{c_1, c_2, c_3, c_4\}$
3. $q = \{2, 1, 2, 2\}$
4. Les préférences de étudiants :

$P_{s_1} = P_{s_4} = P_{s_5} = P_{s_6}$	$P_{s_2} = P_{s_3}$
c_1	c_2
c_2	c_1
c_3	c_3
c_4	c_4

TABLE 2.9 – Exemple 2.4.1, préférences des étudiants

5. $T = \{\tau_1, \tau_2, \tau_3\}$
6. $f(s_1) = f(s_3) = \tau_1, f(s_2) = f(s_5) = \tau_2, f(s_4) = f(s_6) = \tau_3$
7. Les préférences des écoles sont identiques, $\forall c \in C, s_3 \succ_c s_5 \succ_c s_1 \succ_c s_2 \succ_c s_4 \succ_c s_6$

8. Les seules contraintes sont : $\underline{q}_{c_1}^{\tau_3} = 1, \overline{q}_{c_3}^{\tau_3} = \overline{q}_{c_3}^{\tau_2} = 0$

Supposons que l'ordre des étudiants soit celui donné plus haut.

Appliquons la première phase du CDAAI, sans détails, voici les différents *matching* provisoires :

$$\begin{array}{cccccc} \nu_1 & c_1 & c_2 & c_3 & c_4 \\ & s_1 & & & \end{array}$$

$$\begin{array}{cccccc} \nu_2 & c_1 & c_2 & c_3 & c_4 \\ & s_1 & s_2 & & \end{array}$$

$$\begin{array}{cccccc} \nu_3 & c_1 & c_2 & c_3 & c_4 \\ & s_3 & s_2 & & \end{array}$$

$$\begin{array}{cccccc} \nu_4 & c_1 & c_2 & c_3 & c_4 \\ & s_3 & s_2 & s_1 & \end{array}$$

$$\begin{array}{cccccc} \nu_5 & c_1 & c_2 & c_3 & c_4 \\ & \{s_3, s_4\} & s_2 & s_1 & \end{array}$$

$$\begin{array}{cccccc} \nu_6 & c_1 & c_2 & c_3 & c_4 \\ & \{s_3, s_4\} & s_5 & s_1 & \end{array}$$

$$\begin{array}{cccccc} \nu_6 & c_1 & c_2 & c_3 & c_4 \\ & \{s_3, s_4\} & s_5 & s_1 & s_2 \end{array}$$

$$\begin{array}{cccccc} \mu & c_1 & c_2 & c_3 & c_4 \\ & \{s_3, s_4\} & s_5 & s_1 & \{s_2, s_6\} \end{array}$$

A présent, vérifions les cycles.

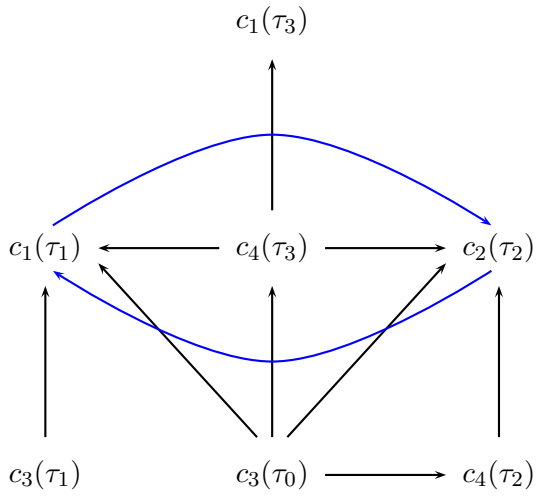

FIGURE 2.1 – Cycles de l'algorithme CDAAI

On a donc un cycle $c_1(\tau_1) \to c_2(\tau_2) \to c_1(\tau_1)$. Dès lors on peut échanger les élèves s_3 et s_5. Le nouveau matching est :

$$\begin{array}{cccc} \mu^* & c_1 & c_2 & c_3 & c_4 \\ & \{s_5, s_4\} & s_3 & s_1 & \{s_2, s_6\} \end{array}$$

Si l'on ré-applique la phase 2 à notre nouveau matching μ^*, on ne trouvera plus de cycle. Donc ce matching est le résultat final du CDAAI.

Ces résultats sont plutôt positifs, du côté stabilité et efficience mais malheureusement, au niveau de la révélation des vraies préférences, on rencontre encore des problèmes.

Théorème 2.4.6.

Pour n'importe quel problème de controlled school choice, il n'existe aucun matching qui soit à la fois stable entre les types, constrained non-wasteful et strategy-proof.

La démonstration de ce théorème se trouve dans [Ehl10].

Dès lors, on peut dire que le *controlled school choice* n'est pas équivalent au simple *many-to-one matching* dans le cas des écoles puisqu'il n'existe pas de *matching* à la fois stable, *constrained non-wasteful* et *strategy-proof*. Le rajout de quotas sur les types des étudiants entraine la perte de la révélation des vraies préférences. Cela peut causer des problèmes, suivant qui va ou ne va pas modifier son comportement, mais nous en parlerons dans la Section 2.5. Le fait que l'algorithme soit séquentiel joue un rôle dans l'assignation des étudiants. L'ordre de classement des étudiants au départ a de l'influence sur le résultat final, cela entraine la perte de l'unicité de la solution.
En se basant sur les algorithmes SDAA et CDAAI, on peut établir la liste des différences qui apparaissent lors de l'introduction des contraintes.

Différences entre le CDAAI et le SDAA :

Algorithme / Propriétés	DAA	CDAAI
Contraintes	Sans	Quotas sur les types
Propositions	Simultanées	Séquentielles
Etudiants	Tous égaux	Egaux entre les différents types
Allocation A chaque étape	Acceptable pour les écoles	Acceptable Peut devenir permanente
Matching	Stable	Stable entre les types et *constrained non-wasteful*
DSCI	Oui	Non
Efficience	S-Optimal pour les étudiants	S-Optimal parmi les *matching* stables entre les types

TABLE 2.10 – Différences entre le CDAAI et le SDAA

Cependant, respecter scrupuleusement les quotas entraîne que le *matching* peut être forcé, en dépit des préférences des étudiants. L'objectif recherché est quand même de maximiser le

bien-être des participants, pas de les placer contre leur gré dans une école. C'est pourquoi nous allons à présent considérer l'approche *Soft bounds*.

Soft bounds

Les quotas ne sont plus des contraintes strictes mais plutôt des lignes de conduites : les priorités deviennent dynamiques et s'adaptent en fonction de la situation. Les étudiants de type τ_i dont le quota minimal n'est pas atteint se voient attribuer une priorité plus importante que ceux dont le quota maximal est dépassé, qui ont une priorité minimale. Enfin, ceux dont le quota minimal est rempli mais pas celui maximal se voient donner une priorité moyenne.
Donc il se peut que les quotas minimaux ne soient pas remplis tant que personne, avec une priorité maximale, ne vient réclamer une place dans cette école.
Ici encore, la notion de stabilité doit être adaptée ainsi que celle de non-wastefulness.

Définition 2.4.6. *Non-wastefulness under soft bounds*

Un matching μ est non-wasteful under soft bounds si
$\forall s \in S, \forall c \in C, cP_s\mu(s)$ *implique que* $|\mu(c)| = q_c$.

Les quotas ne doivent plus être absolument respectés, contrairement à la méthode *Hard bounds* (Définition 2.4.2).

Définition 2.4.7. *Stabilité selon les soft bounds*

Un matching μ est stable selon les soft bounds si
$\forall s \in S, \forall c \in C : cP_s\mu(s)$, *avec* $f(s) = \tau_i$, *on a* $|\mu^{\tau_i}(c)| \geq \underline{q}_c^{\tau_i}$ *et* $s' \succ_c s, \forall s' \in \mu^{\tau_i}(c)$.

Et on a soit :

1. $|\mu^{\tau_i}(c)| \geq \overline{q}_c^{\tau_i}$ *et* $s' \succ_c s, \forall s' \in \mu(c)$ *tel que* $|\mu^{f(s')}(c)| \geq \overline{q}_c^{f(s')}$;
   ou
2. $\overline{q}_c^{\tau_i} > |\mu^{\tau_i}(c)| \geq \underline{q}_c^{\tau_i}$, *et*
   - $|\mu^{\tau_j}(c)| \leq \overline{q}_c^{\tau_j}, \forall j \in \{1, .., k\} \setminus \{i\}$; *et*
   - $s' \succ_c s, \forall s' \in \mu(c)$, *tel que* $\overline{q}_c^{f(s')} > |\mu^{f(s')}(c)| \geq \underline{q}_c^{\tau_i}$.

Les deux définitions ci-dessus sont données par L. Ehlers dans [EHYY12].

Il existe un algorithme qui applique la méthode des soft bounds.

Deferred Acceptance Algorithm with Soft Bounds (DAASB)

Quelques notations sont nécessaires avant de voir l'algorithme.
Soit $S' \subseteq S$ alors $Ch_c(S', q_c, (q_c^{\tau_i})_{i \in \{1,..,k\}})$ est l'ensemble des étudiants $\tilde{S'} \subseteq S'$ qui sont les mieux cotés dans S', suivant les préférences de c, et qui sont inférieurs en nombre à q_c et par type τ_i, inférieurs à $q_c^{\tau_i}$.

De plus,

$Ch_c^{(1)}(\tilde{S}) \equiv Ch_c(\tilde{S}, q_c, (\underline{q}_c^{\tau_i})_{i\in\{1,..,k\}})$,
$Ch_c^{(2)}(\tilde{S}) \equiv Ch_c(\tilde{S} \setminus Ch_c^{(1)}(\tilde{S}), q_c - |Ch_c^{(1)}(\tilde{S})|, (\overline{q}_c^{\tau_i} - \underline{q}_c^{\tau_i})_{i\in\{1,..,k\}})$,
$Ch_c^{(3)}(\tilde{S}) \equiv Ch_c(\tilde{S} \setminus (Ch_c^{(1)}(\tilde{S}) \cup Ch_c^{(2)}(\tilde{S})), q_c - |Ch_c^{(1)}(\tilde{S}) \cup Ch_c^{(2)}(\tilde{S})|, (q_c - \overline{q}_c^{\tau_i})_{i\in\{1,..,k\}})$,
$Ch_c^{(1)}(\tilde{S})$ est l'ensemble des étudiants qui possèdent la plus haute priorité dans $\tilde{S}$ et qui n'excèdent pas les quotas minimaux pour chaque type. $Ch_c^{(2)}(\tilde{S})$ est l'ensemble des étudiants qui possèdent la plus haute priorité dans $\tilde{S}$ et qui n'excèdent pas les quotas maximaux pour chaque type et $Ch_c^{(3)}(\tilde{S})$ est l'ensemble des étudiants choisis au dessus des quotas. Enfin, on a $Ch_c(\tilde{S}) \equiv Ch_c^{(1)}(\tilde{S}) \cup Ch_c^{(2)}(\tilde{S}) \cup Ch_c^{(3)}(\tilde{S})$.

1. A la première étape, chaque étudiant postule dans l'école qui est son premier choix. Soit $S_{c,1}$, l'ensemble des étudiants qui proposent à l'école c à la première étape. L'école c accepte les élèves dans $Ch_c(S_{c,1})$ et rejette le reste.
2. A l'étape $k \geq 2$, les participants qui ont été rejetés à l'étape $k-1$ postulent dans l'école suivante de leur liste. Soit $S_{c,k}$, l'ensemble des élèves qui font une proposition à l'école c à l'étape k et ceux qui sont provisoirement assignés à c, à l'étape $k-1$. Chaque école accepte les étudiants appartenant à $Ch_c(S_{c,k})$ et rejette les autres ;
3. Lorsque chaque élève a une place dans une école ou a été rejetè par toutes les écoles de sa liste, l'algorithme s'arrête.

Il n'est pas évident de comprendre ce qui se passe derrière le choix de $Ch_c(\tilde{S})$ mais c'est assez simple en vérité. A chaque étape, les écoles acceptent les élèves en regardant d'abord si leur quota est rempli ou non, si non, ils sont pris d'office et si leur quota est rempli, leur acceptation dépend des autres élèves postulants et des préférences de l'école. Nous développons un cas particulier de cet algorithme dans le Chapitre 5.

Quelles sont les propriétés de cet algorithme ?

Théorème 2.4.7.

Pour n'importe quel problème de controlled school choice, le DAASB produit un matching stable selon les soft bounds (Définition 2.4.7) et non-wastefull under soft bounds (Définition 2.4.6). De plus, parmi l'ensemble des matching stables et non-wasteful, le DAASB donne le matching optimal pour les étudiants.

Donc dans ces conditions, les étudiants sont 'faiblement' mieux que dans le cas de l'approche *hard bounds*. En effet, la stabilité est plus forte et le critère de *non-wastefulness* moins contraignant. L'objectif recherché est atteint, réduire le poids des quotas et faire passer les préférences des élèves avant tout, cela paye en efficacité.

Qu'en est-il à présent de la stratégie adoptée par les étudiants ?
Le DAASB satisfait une version plus forte que la *strategy-proofness*, il donne un *matching* qui est *group-strategy-proof*.

Définition 2.4.8. *Group Incentive compatible*

Un mécanisme ϕ est group incentive compatible si $\forall \hat{S} \subseteq S, \forall P_S$ (préférences de S), $\nexists P'_S$ tel que $\phi(P'_{\hat{s}}, P_{S\setminus\hat{S}}) P_s \phi(P_S), \forall s \in \hat{S}$. Donc Si les membres d'un groupe changent leurs préférences, cela n'améliore pas l'utilité des membres du groupe.

Cette définition est donnée pas L. Elhers dans [EHYY12].

Théorème 2.4.8.

Le DAASB est group incentive compatible.

Le résultat du DAASB est plus attractif, plus efficient que celui de CDAA mais les quotas ne sont pas toujours respectés. Dès lors, on peut se poser la question : à quoi sert la loi ? Celle-ci est respectée seulement si les étudiants en ont envie, c'est une sorte d'autorégulation par les participants eux-mêmes. Dans ce cas, on peut se poser la question de l'utilité réelle des quotas.

La littérature s'arrête ici, mais avant d'appliquer la théorie du *matching* au cas belge, voici un résumé de ce que nous avons vu jusqu'à présent.

2.5 Résumé et regard critique sur le matching

La littérature concernant le *matching* est bien fournie et ce qui a été présenté plus avant n'est qu'un petit aperçu permettant de comprendre la suite du travail. Avant de passer directement au cas de l'allocation des élèves belges dans les écoles, il a fallu commencer par la base, c'est-à-dire le *one-to-one matching*. Dans ce contexte, nous avons vu l'algorithme de *deferred acceptance* ainsi que ses propriétés de stabilité et de *strategy-proofness*. Le cas des écoles et du *many-to-one matching* n'est pas beaucoup plus compliqué, si ce n'est que plusieurs participants peuvent être associés à une seule école et qu'il faut modifier l'algorithme de DAA. Ensuite, afin de pallier à des déséquilibres de tous types, aussi bien ethniques que socio-économiques, l'Etat introduit des quotas, et les étudiants sont classés par types. Il existe deux approches qui traitent de ce problème. Une méthode consiste à suivre les contraintes scrupuleusement même si cela implique une perte au niveau de la stabilité et une perte d'efficience. Une seconde méthode consiste à donner des priorités plus élevées aux élèves dont les quotas ne sont pas atteints. Cette façon de procéder favorise les préférences des élèves mais il se pourrait qu'elle ne respecte pas les quotas.
La modélisation du *matching* que nous avons abordée jusqu'à présent, comporte des lacunes.

Le modèle proposé ne tient pas compte de l'intensité des choix. Les préférences sont transitives mais la satisfaction obtenue lorsqu'une école est attribuée à un élève dépend de celui-ci et le fait que chacun réagisse différemment n'est pas pris en compte. Ce n'est pas parce que 2 élèves obtiennent leur second choix, qu'ils éprouvent la même satisfaction. Il y a donc une faille dans le fait de modéliser toutes les préférences avec le même niveau de bien-être. Il faudrait normalement prendre en considération l'utilité qu'apporte une école à chaque élève, en plus de sa place dans sa liste. Cependant, il n'est pas évident de quantifier cette utilité, et cela deviendrait vite lourd à manipuler. Par la suite, nous faisons l'hypothèse que le bien-être des élèves, par rapport à une école, est proportionnel à la place qu'occupe cette école dans la liste de leurs préférences (chaque élève aura donc la même fonction d'utilité, cfr.

Section 5.4). Cette hypothèse est assez restrictive et simplificatrice, rendant l'utilité subjective.

La *strategy-proofness* est une propriété attrayante car elle permet de donner des conseils simples aux participants d'un jeu : donner ses vraies préférences est toujours la stratégie gagnante. On suppose toujours que les participants connaissent parfaitement le jeu et peuvent manipuler leurs choix en fonction du mécanisme appliqué. Or en réalité, on a bien souvent affaire à des individus inexpérimentés et d'autres qui le sont plus. De plus manipuler ses préférences est un risque, que tous les participants ne sont pas prêts à prendre. Il faut se rendre compte que nous avons toujours considéré le cas théorique, et que l'équilibre trouvé peut différer de la réalité. Quels sont les participants qui vont ou ne vont pas manipuler leurs préférences ? Si l'on suppose que peu de personnes changent leurs préférences, quelle importance doit-on donner à la *strategy-proofness* dans les algorithmes ?

Les contraintes étant introduites par l'Etat ou toute autre institution pour contrôler les écoles, celles-ci n'ont d'autres choix que de s'y plier. Autrement dit les quotas suppriment le libre arbitre des écoles.

Les quotas, un bien pour un mal ? Si les quotas sont introduits pour diminuer les injustices, paradoxalement ils en créent de nouvelles. Tous les étudiants sont classés par types et ensuite répartis entre les écoles. Dans l'approche *hard bounds*, le fait que les étudiants ne puissent vouloir la place que des personnes du même type est en lui-même une inégalité. Alors que dans l'approche *soft bounds*, les préférences des élèves ont plus de poids que les quotas. Soit, les quotas introduisent de nouvelles inégalités soit, ils ne sont pas vraiment respectés. De plus, on peut se poser la question de qui décide des quotas et quels sont les critères de base pour les élaborer. Est-ce toujours 'juste' ? Pour quels participants est-ce juste ? Dans ce genre de problème, il y a toujours quelqu'un qui y trouve à redire, que ce soient les élèves ou les écoles, certains seront lésés. Lorsqu'un marché est régulé par une institution extérieure, il y a toujours une perte d'efficacité. Il faut donc se poser la question de ce qui est le plus important : équilibrer les écoles avec des quotas imposés ou bien faire passer les préférences des élèves avant tout. Si l'on adopte un point de vue libéral, le mieux est de ne pas intervenir dans le système.

La suite du travail consistera à modéliser le cas belge et à appliquer les algorithmes vus dans cette première partie. Dans un deuxième temps, des algorithmes heuristiques seront élaborés et testés sur notre problème. Le but est de mettre en évidence les propriétés de *strategy-proofness* et de stabilité mais également la rapidité et la robustesse des algorithmes traités.

Chapitre 3

Algorithmes Génétiques

"In the struggle for survival, the fittest win out at the expense of their rivals because they succeed in adapting themselves best to their environment."
Charles Darwin

3.1 Motivations

Un algorithme génétique est un algorithme basé sur les principes de la génétique évolutive, comme son nom l'indique. Son but est de trouver des individus qui sont le mieux adaptés à leur milieu de vie. Cet algorithme d'optimisation peut être appliqué à une classe très large de problèmes. Par exemple, les algorithmes génétiques peuvent optimiser aussi bien des fonctions discrètes, que des fonctions sans dérivées. En fait, ces algorithmes exploitent seulement la manière dont le problème est codé, sans avoir besoin de la dérivée de la fonction à optimiser. Dès lors, ils sont applicables à beaucoup de problèmes.

Pour mieux comprendre comment ils fonctionnent, nous allons détailler toutes les étapes de l'algorithme et les expliciter sur un exemple facile : maximisation de la fonction $f(x) = -\frac{1}{4}x^2 + 2x + 5$, sur $[0, 10]$.

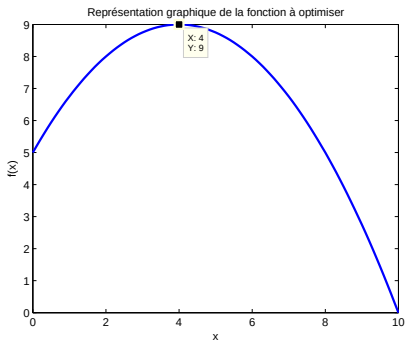

FIGURE 3.1 – Représentation de la fonction à optimiser $f(x) = -\frac{1}{4}x^2 + 2x + 5$, sur $[0, 10]$.

Remarque : le terme algorithme génétique sera parfois abrégé par GA.

3.2 Création de la population

Les algorithmes génétiques s'inspirent de la biologie et dès lors nous retrouvons des termes utilisés en génétique. Voici une définition du terme chromosome, tirée du site Trésor de la Langue Française [TFL].

Définition 3.2.1. *Chromosome*

Élément du noyau cellulaire, de forme caractéristique et en nombre constant pour une espèce donnée, et considéré comme le support des facteurs héréditaires.

A présent, nous pouvons décrire les composantes des chromosomes.

1. Chaque chromosome porte des **gènes** qui contiennent chacun une partie de l'information génétique (séquence ADN). Ce sont les gènes qui déterminent les caractéristiques des êtres vivants, tels que la couleur des yeux ;
2. La valeur que les gènes peuvent prendre est appelée **allèle**. Cela correspond à b pour la couleur bleue, et B pour brun dans le cas de la couleur des yeux ;
3. Le **phénotype** est l'expression (visible) des gènes. Il faut savoir qu'il existe 2 types d'individus : les haploïdes et les diploïdes, les premiers ne possèdent qu'un seul exemplaire de chaque chromosome alors que les seconds en possèdent une paire. Dans le cas des premiers le phénotype est directement déterminé par l'allèle, dans notre exemple, un individu portant l'allèle b aura les yeux bleus et un autre ayant B aura des yeux bruns. Mais pour les seconds, les allèles sont soit dominants soit récessifs. Pour que le caractère d'un gène récessif s'exprime, il doit être présent deux fois, sinon, c'est l'allèle dominant qui s'exprime. Par exemple si b est récessif et B dominant, alors un individu qui possède la paire bb aura les yeux bleus, bB les yeux bruns, Bb les yeux bruns et finalement BB les yeux bruns. Dans la suite, tous les individus sont haploïdes ;
4. La place d'un allèle dans un chromosome est le **locus** ;
5. Un ensemble de chromosomes forme la **population** sur laquelle l'algorithme génétique va travailler.

Pour créer la population de départ, il faut discrétiser l'espace des solutions (un chromosome est représenté par un vecteur et correspond à une solution possible). Le plus souvent, on utilise une représentation binaire. Une fois le modèle réalisé, on tire au hasard notre population de départ dans l'espace des solutions.

Dans notre exemple, l'espace des solutions est l'intervalle $[0, 10]$ et chaque solution x est envoyée sur un nombre entre 0 et 1000 afin de représenter facilement un nombre à trois décimales. Ce nombre est lui traduit en binaire dans un vecteur de 10 bits. Voici ce que cela donne pour une population de 5 solutions.

Chromosome	valeur$_{10}$	x(phénotype)
0001101001	105	1.05
1111000010	962	9.62
0011111011	255	2.55
1110001011	907	9.07
1101110111	887	8.87

TABLE 3.1 – Tableau de la population de départ, extrait de [Dal13]

Remarque : par abus de langage, nous utiliserons aussi bien les termes solution, individu et chromosome pour désigner une solution dans l'espace des solutions.

3.3 *Fitness*

A chaque individu de la population, on assigne une valeur qui correspond au niveau d'adaptation de l'individu à son environnement. La *fitness* est en lien direct avec la survie et reproduction d'un individu, ou sa disparition de la population. Plus un individu est adapté à son environnement, plus sa *fitness* est élevée et plus il aura de chance de se reproduire. Dans notre exemple, chaque individu x reçoit la valeur de la fonction en x comme *fitness*, puisque l'objectif est la maximisation de la fonction.

3.4 Mécanisme de l'algorithme

Basé sur le phénomène de la sélection naturelle, l'algorithme génétique (AG) est un procédé itératif. En partant de la population de départ et en passant par les 3 étapes ci-dessous, l'algorithme crée une nouvelle population à chaque étape (génération).

- La sélection ;
- Le croisement (*crossover*) ;
- La mutation.

Un algorithme génétique est représenté de la manière suivante :

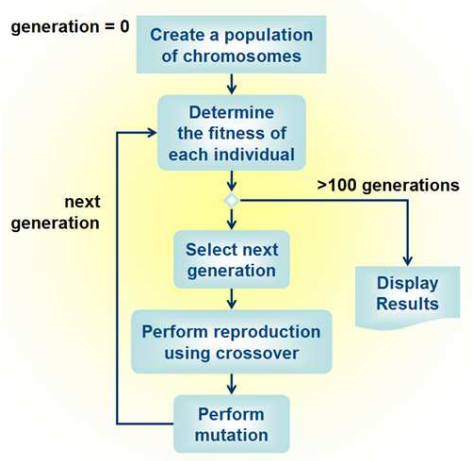

FIGURE 3.2 – Etapes de l'algorithme génétique, image extraite de [Dal13]

La nouvelle population créée à chaque étape conserve les meilleures caractéristiques de l'ancienne génération et ainsi, on espère tendre vers une amélioration de la population de départ et donc atteindre l'objectif d'optimisation.

3.5 Sélection

La première étape est la sélection d'individus susceptibles de se reproduire, c'est-à-dire ceux qui sont le mieux adaptés à leur environnement. On en sélectionne un nombre fixe n durant tout l'algorithme. Ces individus forment le *mating pool*. Cet échantillon peut être sélectionné selon différentes méthodes, dont la plus fréquente est celle de la roulette.

Roulette *wheel*

La *fitness* de chaque individu dans la population est sommée ($=F$). Ensuite, pour chaque individu, on calcule le pourcentage p_i de sa *fitness* par rapport à F. On place ces pourcentages sur une roulette et pour sélectionner un individu, on fait simplement tourner la roulette. Dès lors, chaque individu a une probabilité p_i d'être sélectionné.
Dans notre exemple, cela correspond au tableau suivant :

Chromosome	valeur$_{10}$	x(phénotype)	$f(x)$	Pourcentage
0001101001	105	1.05	6.82	31
1111000010	962	9.62	1.11	5
0011111011	255	2.55	8.48	38
1110001011	907	9.07	2.57	12
1101110111	887	8.87	3.08	14
		Somme	22.05	100

TABLE 3.2 – Tableau pour la sélection de la population, extrait de [Dal13]

Lequel donne graphiquement :

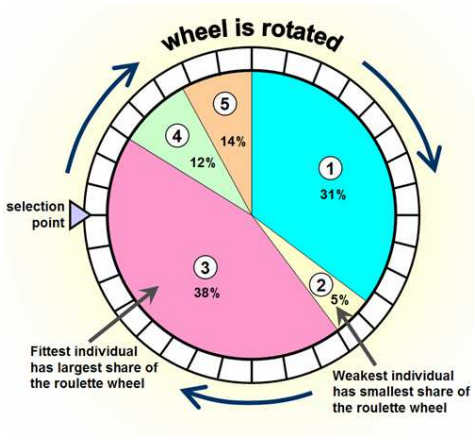

FIGURE 3.3 – Sélection par le procédé de la roulette, image extraite de [Dal13].
Dans l'image, l'individu 3 a le plus de chance d'être sélectionné avec sa *fitness* représentant 38% de la somme des *fitness* alors que l'individu 2 n'a quasi aucune chance d'être sélectionné.

Dans cette sélection, les individus qui ont une *fitness* faible ont peu de probabilité d'être sélectionnés. Le risque est de sélectionner le même individu plusieurs fois et donc de biaiser la population. On n'explore pas toutes les possibilités de l'espace et la population converge trop

vite vers un seul individu. Or un intérêt des algorithmes génétiques est d'explorer facilement l'espace des solutions afin d'éviter les minima locaux. C'est pourquoi, la méthode suivante est une alternative à la roulette *wheel*.

Remainder Stochastic

Pour chaque individu, à partir du pourcentage calculé comme dans la roulette *wheel*, on calcule la valeur $e_i = p_i \times n$, où n est le nombre d'individus sélectionnés. Dans notre exemple, si on sélectionne 5 individus, on obtient :

Chromosome	valeur$_{10}$	x(phénotype)	$f(x)$	Pourcentage	e_i
0001101001	105	1.05	6.82	31	1.55
1111000010	962	9.62	1.11	5	0.25
0011111011	255	2.55	8.48	38	1.90
1110001011	907	9.07	2.57	12	0.6
1101110111	887	8.87	3.08	14	0.7
		Somme	22.05	100	5

TABLE 3.3 – Tableau pour la sélection de la population (remaider stochastic), extrait de [Dal13]

Ensuite, les individus sont triés de manière décroissante en fonction de leur e_i. On regarde la partie entière des e_i et on prend un nombre égal à cette partie entière d'individu i (pour le *mating pool*).

On prend une fois les individus 1 et 3.

Enfin, pour les places restantes, chaque individu est passé en revue et pour chacun, on effectue une épreuve de Bernoulli avec la partie fractionnelle des e_i comme probabilité de succès.

Dans notre exemple, on peut supposer que l'individu 3 est pris une seconde fois et que les individus 5 et 2 sont sélectionnés une fois.

Ainsi, les individus qui ont une faible probabilité ont une chance de se retrouver dans le *mating pool*.

3.6 *Crossover*

Le *crossover* ou reproduction du *mating pool* s'effectue comme suit : on prend 2 par 2 les individus dans le *mating pool*, sans remise, et on les croise avec une probabilité de croisement $p_{crossover}$. Si il n'y a pas de croisement, les individus sont remis tels quels dans la nouvelle population qu'on appellera population croisée. Si le croisement a lieu, il s'effectue de la sorte : on prend un point de coupure au hasard, au même endroit sur les 2 chromosomes de départ (parents). La progéniture se compose de deux nouveaux chromosomes, chacun formé d'une partie de chaque chromosome parent. Le premier enfant a la première partie du parent 1 et la seconde du parent 2 et c'est le contraire pour l'enfant 2. Les deux nouveaux chromosomes rejoignent la population croisée. Cette opération vise la création de nouveaux individus qui possèdent le matériel génétique de chacun de leurs parents. Nous espérons qu'ils seront

meilleurs que ces derniers, en terme de *fitness*.

Dans notre exemple, supposons que les chromosomes 1 et 3 aient été choisis pour la reproduction et prenons le point de coupure après le $4^{ème}$ bit.

0	0	0	1	1	0	1	0	0	1
0	0	1	1	1	1	1	0	1	1
Partie 1				Partie 2					

En échangeant les parties, nous obtenons les enfants :

0	0	0	1	1	1	1	0	1	1
0	0	1	1	1	0	1	0	0	1

L'enfant 1 représente la solution 1.23, ce qui lui fait une *fitness* de 7.08. Cette *fitness* est meilleure que celle du premier parent mais pas que celle du second parent. Au vu des individus de la population de départ, cette nouvelle solution est assez bonne. L'enfant 2 représente la solution 2.33, avec une *fitness* de 8.30, ce qui en fait un très bon individu.

3.7 Mutation

L'étape suivante est la mutation des individus. Celle-ci fait muter les chromosomes afin d'explorer tout l'espace des solutions et de ne pas converger trop vite vers un individu. Elle s'effectue sur la population de départ pour ne pas abîmer les individus créés à partir du *crossover*. La probabilité de mutation de départ est fixée à $p_{mutation}$. Cela signifie que chaque gène a une probabilité $p_{mutation}$ d'être muté où encore que la probabilité qu'un gène soit muté suit une loi de Bernoulli de paramètre $p = p_{mutation}$. Cependant, passer en revue chaque gène prend un temps considérable, alors que le nombre total de mutations suit une loi binomiale de paramètres $N = Taillepop \times Lmatching$ et $p_{mutation}$, où $Taillepop$ est la taille de la population de départ et $Lmatching$ la longueur du *matching*. On peut donc calculer le nombre moyen total de mutations :

$$n_{mutation} = \text{Bi}(Taillepop \times Lmatching, p_{mutation})$$

Avec $n_{mutation}$ le nombre total de mutations.

Il suffit de choisir $n_{mutation}$ fois un chromosome au hasard dans la population et de muter un de ses gènes au hasard.

En binaire, muter un gène revient à transformer sa valeur actuelle : un 0 est remplacé par un 1 et vice versa.

Dans notre exemple, prenons le chromosome 2 et mutons son premier bit (gène).

1	1	1	1	0	0	0	0	1	0
↑									

Il devient donc :

0 1 1 1 0 0 0 0 1 0

Ce nouvel individu représente la solution 4.50, avec a une *fitness* de 8.93, ce qui prouve qu'à partir d'un individu médiocre, on peut trouver un très bon individu, en parcourant l'espace des solutions.

L'ensemble des individus mutés forment la population mutée.

3.8 Génération

Après avoir effectué les 3 étapes décrites ci-dessus, nous avons 3 populations : la population de départ, la population croisée et la population mutée. On applique l'opérateur sélection à ces populations pour trouver les meilleurs individus parmi ces populations, ceux-ci forment la nouvelle génération. On peut ainsi recommencer le processus (reproduction, mutation, sélection), avec comme population de départ la nouvelle génération.
L'algorithme s'arrête après un nombre fini de générations et produit une population finale qui l'on espère est mieux adaptée à son environnement. On peut également poser comme critère d'arrêt un seuil d'adaptation à atteindre.

Dans note exemple, les x vont converger vers 4, avec $f(4) = 9$, le maximum de la fonction f.

Remarque : même si l'on prend une population de départ identique, il est probable que l'algorithme génétique ne donnera pas la même population finale car c'est un processus stochastique (sauf dans le cas de problèmes simples comme l'exemple utilisé).

Les informations concernant les algorithmes génétiques ont été tirées de [Mit98] et de [Gol89].

3.9 Algorithme multi-objectif

Lorsqu'on veut optimiser plusieurs fonctions objectifs en même temps, nous avons deux approches possibles. La première, est une approche a priori, qui consiste à définir des poids pour chaque fonction objectif afin de combiner celles-ci et d'optimiser la nouvelle fonction, de la manière classique décrite ci-dessus. La seconde, est une approche a posteriori qui utilise les courbes de Pareto-optimalité. Cette dernière est également appelée optimisation multi-objective.

Cette optimisation nécessite de modifier quelque peu l'algorithme génétique de base. Ce qui change est la notion de *fitness*. En effet, désormais la *fitness* a plusieurs composantes.
Comme la sélection s'effectue en fonction d'une *fitness* unique, il faut donner une *fitness* fictive (à une seule composante) aux chromosomes, celle-ci sera déterminée en fonction des composantes de la *fitness* effective.

Tri

Tout d'abord, nous allons trier les individus selon leur *fitness* (ordre décroissant). Cependant, nous devons nous baser sur toutes les composantes de cette *fitness*, c'est pourquoi nous introduisons la définition suivante.

Définition 3.9.1. *Ensemble Pareto-optimal*

Soit deux solutions, x_1 et x_2, il y a deux possibilités : l'une domine l'autre ou aucune des deux ne domine l'autre.
La solution x_1 domine x_2 si :

1. *x_1 est au moins aussi bonne que x_2 dans tous les objectifs. C'est-à-dire, $f_j(x_1) \geq f_j(x_2), \forall j$, avec j, les différents objectifs ;*
2. *x_1 est strictement meilleure que x_2 dans au moins 1 objectif, $\exists j : f_j(x_1) > f_j(x_2)$.*

Soit P, un ensemble de solutions, s'il n'existe aucune solution, dans l'espace de recherche, qui domine au moins un membre de P, alors les solutions de l'ensemble P forment un ensemble Pareto-optimal global.

Remarque : lorsqu'on ne considère pas toutes les solutions de l'espace mais seulement une partie, nous parlerons d'ensemble non dominé et non d'ensemble Pareto-optimal.

Maintenant que nous avons défini un ensemble non dominé, pour trier toutes nos solutions, il suffit de prendre l'ensemble non dominé de l'ensemble des solutions, qui sera le front de Pareto-optimalité (solutions de rang 1). Ensuite, on l'enlève de l'ensemble des solutions et on recommence pour trouver l'ensemble des solutions de rang 2 et ainsi de suite jusqu'à ce que toutes les solutions soient dans un ensemble.
Ainsi, nous avons trié les solutions.

Par exemple, cette définition est beaucoup utilisée en microéconomie, pour classer les préférences des individus.

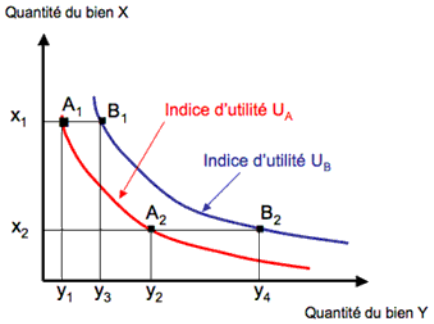

FIGURE 3.4 – Front de Pareto-optimalité : exemple pour les préférences des individus, extraite de http://public.iutenligne.net/economie/Simonnet/consommateur/docs/indifference2.html

Les points situés sur la courbe U_B dominent ceux de la courbe U_A.

Il existe différentes fonctions qui implémentent cette procédure, consultables dans [Deb03] mais nous avons décidé d'implémenter nous-même notre propre fonction tri.

Supposons, que la *fitness* est composée de 2 fonctions objectifs, f_1 et f_2.

1. D'abord, nous utilisons la fonction tri pour trier les individus selon la première composante de leur *fitness* (ordre décroissant) ;
2. Nous prenons les individus qui ont leur première fonction objectif égale à f_{1max} (valeur maximale de la première composante f_1) ;
3. Nous trions les individus de l'étape précédente en fonction de la seconde composante de leur *fitness*. Nous prenons pour le rang 1, tous les individus dont la seconde fonction objectif est égale à f_{2max} (valeur maximale de la seconde composante f_2). Les autres individus sont rejetés et feront partie d'un rang supérieur ;
4. Ensuite, nous prenons les individus qui ont la plus grande fonction objectif 1, parmi ceux restants. Encore une fois, on trie ces individus selon la seconde composante de la *fitness*. Si la seconde composante maximale de ces individus est plus grande que f_{2max}, alors elle devient f_{2max} et les individus ayant comme seconde fonction objectif cette valeur sont pris dans le rang 1. Les autres individus sont rejetés. On recommence cette étape jusqu'à ce qu'il n'y ait plus aucun individu dans la liste de départ, alors le rang 1 est complet. On recommence à l'étape 2 avec les individus qui avaient été rejetés pour le rang 1 et on forme le rang 2 ;
5. Ces étapes se répètent jusqu'à ce que tous les individus fassent partie d'un rang.

La généralisation de cette procédure à plus de 2 fonctions objectifs est laissée libre au lecteur.

A présent, nous devons attribuer une *fitness* fictive à chaque solution pour ramener le problème à une optimisation en une dimension.

Fitness

Le but de l'algorithme multi-objectif est de parcourir l'espace des solutions, en maintenant la diversité des solutions. Dès lors, il faut être prudent quant à la *fitness* fictive que nous allons attribuer aux chromosomes.
Les individus qui sont trop proches les uns des autres doivent être pénalisés car nous ne voulons pas que l'algorithme converge trop rapidement vers quelques individus. C'est pourquoi, nous allons utiliser la méthode de la *sharing function* :

Supposons que toutes les solutions de rang 1 aient une *fitness* fictive de départ f_1, à partir de la procédure suivante, on peut assigner à chaque individu, une *fitness* qui tient compte de la proximité des autres solutions.
Soit l'ensemble des n_k solutions du $k^{ème}$ front d'optimalité,

1. Pour chaque individu i du front, on calcul sa distance euclidienne normalisée par rapport à chaque autre individu j de l'ensemble :

$$d_{ij} = \sqrt{\sum_{p=1}^{P} \left(\frac{x_i^p - x_j^p}{x_u^p - x_l^p} \right)^2}$$

   Avec x_i^p, la $p^{ème}$ composante de la solution x_i [1], x_u^p, la valeur maximale que peut prendre la $p^{ème}$ composante et x_l^p, la valeur minimale et P est le nombre de composantes des solutions ;

2. Comparaison de la distance d_{ij} à un paramètre σ_{share}, déterminé après :

$$Sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{share}} \right)^2 & \text{si } d_{ij} \leq \sigma_{share} \\ 0 & \text{sinon.} \end{cases} \qquad \forall w \in W$$

3. Si $j < n_k$, on incrémente j et on retourne à l'étape 1, sinon on calcule

$$m_i = \sum_{j=1}^{n_k} sh(d_{ij})$$

4. Ajustement de la *fitness* de l'individu i

$$f_i' = \frac{f_k}{m_i}$$

5. Cette méthode est appliquée à chaque individu du front
6. Lorsque ceux-ci ont tous leur *fitness* fictive, on prend

$$f_{k+1} \leq \min_{i \in \{0..n_k\}} f_i'$$

   Ainsi, nous pouvons donner une fitness f_i' aux individus de chaque front.

Le paramètre σ_{share} est calculé de la manière suivante :

$$\sigma_{share} \approx \frac{0.5}{\sqrt[P]{q}}$$

Où q est le nombre de solutions Pareto-optimales distinctes désirées. Dans beaucoup de problèmes, $q \approx 10$ et cette approximation trouve en général de bonnes solutions.

Les informations sur les algorithmes génétiques mutli-objectifs sont tirées de [Deb03] et de [Seh06].

1. Il existe également une variante où les composantes de la solutions x_i sont remplacées par les composantes de la *fitness* correspondante.

3.10 Avantages et désavantages

Les avantages des algorithmes génétiques par rapport aux méthodes d'optimisation comme la méthode du gradient, les méthodes énumératives, etc, sont :

- Ils travaillent avec les paramètres de la fonction à optimiser, pas celle-ci directement. De ce fait, ils peuvent optimiser des fonctions sans dérivée explicite et même des fonctions discrètes. Pour ce faire, ils utilisent la notion de *fitness* ;
- Ils partent d'un ensemble de points et non pas simplement d'un seul individu donc ils parcourent l'espace des solutions. Les algorithmes génétiques trouvent plusieurs solutions à la fois, ce qui leur permet de ne pas rester bloqués dans des minima locaux ;
- Par rapport aux méthodes d'optimisation classiques (méthode du gradient, Newton, etc), il n'y a aucune accumulation d'erreur due à une approximation du gradient, surtout lorsqu'on travaille en grande dimension.

Il y a une part d'aléatoire dans le processus de construction des solutions, même si l'algorithme tient compte de l'historique des populations. Chaque nouvelle génération est formée de telle sorte qu'elle garde les meilleures caractéristiques de l'ancienne génération.

Le seul désavantage des algorithmes génétiques est qu'il n'existe aucune preuve théorique de leur convergence. Ils convergent en probabilité et les solutions qu'ils obtiennent varient à chaque application de l'algorithme. Dès lors on peut se demander pourquoi les utiliser. Il s'avère que nous observons dans beaucoup de cas une convergence pratique de ce genre d'algorithme. Le tout est de bien coder le problème de départ et de bien adapter les outils décrits ci-dessus. Il est vrai que les GA peuvent ne pas fonctionner du premier coup mais en ajustant les paramètres, il est possible de construire un algorithme efficient, qui donne de bons résultats. Evidemment, pour prouver qu'un résultat est correct, nous devons déduire nos conclusions à partir de la moyenne de plusieurs simulations.
Il y aurait encore beaucoup à dire sur les algorithmes génétiques cependant, le but de ce mémoire n'est pas d'analyser leurs propriétés en détails. Même si elle n'est qu'heuristique, la preuve de l'efficacité des GA n'est plus à démontrer pour de nombreux problèmes mathématiques appliquées. Nous montrons dans la suite, qu'ils peuvent être utiles, dans le contexte de la microéconomie.

Remarque : Nous avons présenté ici les étapes d'un algorithme génétique classique mais il est évident qu'il peut être modifié afin d' être adapté au problème donné. C'est d'ailleurs la partie la plus compliquée, la modélisation d'un problème donné et l'adaptation des étapes de l'algorithme génétique.

Chapitre 4

Application au problème du mariage

"Le mariage est un état trop parfait pour l'imperfection de l'homme."
Chamfort

Le but de ce chapitre est d'appliquer les algorithmes génétiques au problème du mariage, afin de dégager des méthodes qui pourraient fonctionner correctement dans le cadre de notre problème belge. Pour cela, nous avons décidé d'appliquer les algorithmes génétiques au problème du mariage décrit dans la Section 2.1. Pour rappel, le but est de marier des hommes et des femmes. Nous disposons d'un ensemble de femmes W et d'un ensemble d'hommes M ainsi que de leur préférences, respectivement P_W et P_M. Si l'on se réfère à l'algorithme de Gale & Shapley, le DAA, décrit à la section Section 2.1, les individus d'un groupe proposent aux individus de l'autre groupe et ces derniers choisissent leur partenaire. Dans notre exemple, ce sont les hommes qui proposent et les femmes qui disposent. Le problème revient évidemment à maximiser le bien-être (utilité voir Définition 2.1.4) des individus prenant part au jeu. Cette utilité va être utilisée comme la *fitness* des algorithmes génétiques, un choix tout indiqué pour ce problème de maximisation sans dérivée. En effet, nous ne pouvons pas utiliser de méthode d'optimisation classique car celles-ci requièrent des hypothèses sur la fonction à optimiser et sa dérivée (par exemple Lipchitz, dérivée continue, etc), et nous ne possédons pas ces informations dans notre problème (nous avons une fonction discrète à maximiser).

A présent, passons en revue les différentes composantes d'un algorithme génétique, adaptées à notre problème de mariage.

4.1 Création de la population

Pour construire l'algorithme génétique, nous avons besoin de définir nos paramètres et leur équivalence au niveau algorithmique. Nous avons repris les paramètres décrits dans la Section 3.2 et voici ce que cela donne :

- La population est un ensemble de chromosomes ;
- Un chromosome représente un *matching* ;
- Un gène représente un individu du groupe des hommes (qui proposent) ;
- Le locus, ou la place que le gène occupe dans le chromosome, représente un individu du groupe des femmes.

Nous ne pouvons pas coder notre problème en binaire car chaque individu est différent et possède des caractéristiques particulières. Nous avons donc opté pour un langage décimal : chaque homme et chaque femme est représenté par un numéro. Chaque chromosome est un vecteur dont les composantes représentent les hommes et la coordonnée d'un homme dans le vecteur représente la femme qui lui est associée. Si une femme est célibataire, son partenaire est symbolisé par le numéro de cette femme, précédé d'un signe moins. Si un homme est célibataire, il n'apparait tout simplement pas dans le chromosome. Il est évident que si nous utilisions la version de l'algorithme dans laquelle les femmes proposent, les places des individus seraient échangées.

Un petit exemple est plus parlant qu'un long discours.

Exemple 4.1.1. *Prenons un ensemble de 5 femmes et un de 5 hommes :* $W = \{1, 2, 3, 4, 5\}$ *et* $M = \{1, 2, 3, 4, 5\}$.
Soit le chromosome suivant :

4	*2*	*-3*	*1*	*3*

On a que la femme 1 est liée à l'homme 4, la 2 à l'homme 2, la 3 est célibataire, la femme 4 est liée à l'homme 1 et la 5 à l'homme 3. L'homme 5 reste également célibataire puisqu'il n'apparait pas dans le vecteur.

Remarque : par abus de langage, nous utiliserons *matching* pour chromosome et vice versa.

En pratique, pour modéliser nos individus, on tire d'abord au hasard les préférences des femmes et des hommes. Ces préférences sont indépendantes et on suppose que les participants révèlent leurs vraies préférences. Pour chaque individu, on prend au hasard, un nombre fixé de personnes dans le groupe de l'autre sexe.

Ensuite, pour former la population initiale, les chromosomes sont créés aléatoirement et indépendamment les uns des autres. Chaque chromosome est construit de la manière suivante : pour chaque femme, on prend aléatoirement un homme dans la liste de ses préférences. De plus, on vérifie si la femme fait bien partie des préférences de l'homme choisi. Si c'est le cas, alors on les associe, sinon la femme reste célibataire.

Une propriété intéressante des *matching* est la stabilité (au sens de la Définition 2.1.3). Elle permet d'assurer la longévité des couples formés car sans elle, les paires qui bloquent le *matching* (voir Définition 2.1.2) auraient tout intérêt à quitter leur partenaire respectif et à se marier ensemble. Sans la stabilité, un mécanisme qui couple les individus n'a aucun sens. Cependant, lorsque le nombre de femmes et d'hommes augmente, la méthode aléatoire, décrite ci-dessus, donne de moins en moins de *matching* stables et il est difficile de faire évoluer un *matching* instable en un *matching* stable. Pour solutionner ce problème, nous avons décidé que 10% de la population de départ serait composée du *matching* stable le plus simple, à savoir, tout le monde est célibataire.

4.2 *Fitness*

Tout d'abord, il est évident que si plusieurs hommes sont associées à une même femme, la *matching* ainsi formé n'est pas acceptable et la *fitness* est égale à zéro. Mis à part cela, il y a plusieurs moyens de calculer la *fitness* de la population en tenant compte des préférences des femmes ou des hommes ou encore de la stabilité des *matching*, ou bien en combinant ces différents paramètres.

1. Pour calculer la *fitness* d'un chromosome en fonction des femmes, il suffit de regarder l'utilité de chacune d'elles et de prendre la moyenne arithmétique de ces utilités. L'utilité d'un individu dépend de la place de son partenaire dans sa liste de préférences, comme défini par la Définition 2.1.4. Nous avons choisi de représenter l'utilité d'une femme w par la fonction suivante :

$$u_w(\mu(w)) = \begin{cases} 1 - \dfrac{rk_w(\mu(w)) - 1}{|P_w|} & \text{si } \mu(w) \in P_w \\ 0 & \text{sinon.} \end{cases} \qquad \forall w \in W$$

   Avec, $rk_w(\mu(w))$ est la place du partenaire $\mu(w)$ dans la liste de préférences de w et $|P_w|$ le nombre de préférences de w. L'utilité d'une femmes w est la plus élevée lorsqu'elle est couplée avec son partenaire favori m, $rk_w(m) = 1$ et $u_w(m) = 1 - \dfrac{1-1}{|P_w|} = 1$. Par contre, elle sera la plus basse si w est couplée avec elle-même : $rk_w(w) = |P_w|$ (w est le dernier élément de sa liste de préférences) donc $u_w(m) = 1 - \dfrac{|P_w| - 1}{|P_w|} = \dfrac{1}{|P_w|}$.

   Cette représentation de l'utilité des femmes est arbitraire et il existe bien d'autres moyens de transformer des préférences ordinales en préférences cardinales.

   Nous aurons donc que la *fitness* du *matching* μ, pour les femmes, est donnée par :

$$\mathcal{F}_W(\mu) = \frac{\sum_{w \in W} u_w(\mu(w))}{|W|} \tag{4.1}$$

   Avec, $|W|$ le nombre de femmes.

2. Pour calculer l'utilité individuelle d'un homme, la même formule est utilisée.

$$u_m(\mu(m)) = \begin{cases} 1 - \dfrac{rk_m(\mu(m)) - 1}{|P_m|} & \text{si } \mu(m) \in P_m \\ 0 & \text{sinon.} \end{cases} \qquad \forall m \in M$$

   Avec, $rk_m(\mu(m))$ est la place du partenaire $\mu(m)$ dans la liste de préférences de m et $|P_m|$ le nombre de préférences de m. Encore une fois, cette représentation est arbitraire et assez simple.

   Nous aurons donc que la *fitness* du *matching* μ, pour les hommes, est donnée par :

$$\mathcal{F}_M(\mu) = \frac{\sum_{m \in M} u_m(\mu(m))}{|M|} \tag{4.2}$$

Avec, $|M|$ le nombre d'hommes.

3. Si l'on ne désire travailler qu'avec des *matching* stables, au sens de la Définition 2.1.3, on pose la *fitness* des *matching* instable égale à zéro.

Nous pouvons également choisir une combinaison pondérée de ces trois points pour former la *fitness*.

La *fitness* est, entre autres, un des critères qui nous permettra de comparer les algorithmes génétiques à l'algorithme de Gale & Shapley, dans la Section 4.8.

Remarque : par abus de langage, nous utiliserons *fitness* pour utilité et vice versa.

Parmi les chromosomes de départ, il faut sélectionner les meilleurs, qui seront candidats pour le croisement.

4.3 Sélection

La sélection que nous avons choisie est la sélection par *stochastic remaider*, décrite à la Section 3.5.

Lorsque les individus sont sélectionnés, il reste à les croiser deux à deux. Nous avons fait appel à différentes méthodes de *crossover*, les voici plus en détails.

4.4 *Crossover*

La finalité du *crossover* est d'obtenir de meilleurs individus en échangeant certaines parties des chromosomes sélectionnés. Pour effectuer le *crossover*, il suffit de sélectionner deux individus dans la population et de les croiser avec une probabilité de croisement, notée $p_{crossover}$. Pour une question de facilité, les deux chromosomes choisis pour la reproduction sont appelés les parents, et les deux chromosomes qui résultent du croisement sont appelés les enfants.

Crossover classique

La première méthode consiste à prendre deux parents et à les couper chacun en deux (la coupure est située au même endroit dans les deux chromosomes). Ensuite, on échange les parties pour former les enfants. Les deux enfants sont alors constitués d'une partie différente de chaque parent. Il faut faire attention qu'avec cette méthode de *crossover*, on peut rencontrer plusieurs fois le même homme dans un même *matching* et donc celui-ci aura une *fitness* nulle.

Exemple 4.4.1.

Parent1 = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Parent2 = | 1 | 6 | 3 | 5 | 8 | 9 | 4 | 10 | 7 | 2 |

Enfant1 =	*1*	*2*	*3*	*4*	*8*	*9*	*4*	*10*	*7*	*2*

Enfant2 =	*1*	*6*	*3*	*5*	*5*	*6*	*7*	*8*	*9*	*10*

Comme on peut le constater, dans le premier enfant, l'homme 4 est marié à deux femmes différentes ainsi que l'homme 2, et dans le second enfant, le même phénomène se produit avec les hommes 5 et 6. Ces deux enfants ont une *fitness* nulle.

Pour pallier à ces problèmes, nous nous sommes tournés vers des méthodes de *crossover* qui avaient été utilisées dans le cadre du *Traveling Salesman Problem* (TSP). Un représentant de commerce doit passer par certaines villes (une seule fois) et emprunter le chemin le plus court. Ce problème est intéressant car il utilise une représentation décimale (chaque ville est un chiffre) et la contrainte qu'un chiffre ne peut être présent qu'une seule fois par chromosome. Vous trouverez l'explication en détail par I. M. Oliver, D. J. Smith et J. R. C. Holland dans [OSH87], nous ne décrivons que les trois méthodes de *crossover* développées pour le TSP.

Ordercrossover

On coupe chaque parent en deux points. La séquence du milieu du premier parent est recopiée identiquement dans le premier enfant. Ensuite, nous nous servons de l'ordre des éléments du second parent pour remplir l'enfant 1. On commence par la case qui suit le second point de coupure et on y met le premier élément du second parent. On passe à la case suivante dans l'enfant 1 et on y met le second élément du parent 2 et on remplit ainsi de suite le vecteur enfant. Si l'on rencontre dans le parent 2 un élément qui est déjà présent dans l'enfant 1, on passe à l'élément suivant dans le parent 2. Le deuxième enfant est construit symétriquement.

Exemple 4.4.2.

Parent1 =	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*

Parent2 =	*1*	*6*	*3*	*5*	*8*	*9*	*4*	*10*	*7*	*2*

Enfant1 =	*3*	*8*	*9*	*4*	*5*	*6*	*7*	*10*	*2*	*1*

Enfant2 =	*3*	*5*	*6*	*7*	*8*	*9*	*4*	*10*	*1*	*2*

Grâce à cette méthode de reproduction, les positions absolues de certains éléments des deux parents sont conservées et cela favorise l'apparition de nouveaux chromosomes mieux adaptés à leur environnement.

PMX

Cette troisième méthode demande aussi 2 points de coupure, appliqués aux deux parents. Les éléments centraux du premier parent sont conservés et placés dans l'enfant 1. Ensuite, les éléments du second parent sont ajoutés, en conservant leur ordre. Cependant, il faut faire attention de ne pas ajouter les éléments qui sont déjà présents dans la progéniture. Si un chiffre x du parent 2 est déjà présent dans l'enfant, on le remplace par l'élément du parent

2 qui possède le même locus que l'élément x du parent 1. Le second enfant est construit symétriquement.

Exemple 4.4.3.

Parent1 =	1	2	3	4	5	6	7	8	9	10
Parent2 =	1	6	3	5	8	9	4	10	7	2
Enfant1 =	1	9	3	8	5	6	7	10	4	2
Enfant2 =	1	2	3	7	8	9	4	5	6	10

Cette méthode conserve aussi certaines positions absolues de chaque parent et on espère ainsi ne garder que les meilleures caractéristiques de chaque parent.

Cycle

Cette méthode est basée sur les deux conditions suivantes :

- Chaque élément de la progéniture doit être un élément de l'un ou de l'autre parent et avoir le même locus que ce dernier ;
- La progéniture est une permutation.

Exemple 4.4.4. *Considérons les deux parents suivants :*

Parent1 =	1	2	3	4	5	6	7	8	9	10
	↓	↓		↓						
			↑			↑	↑	↑	↑	↑
Parent2 =	2	4	7	1	5	8	10	3	6	9

Le premier élément doit être soit 1 soit 2. Supposons qu'on choisisse l'élément 1, du premier parent. Alors, l'élément 2 (qui a le même locus que l'élément 1 dans le second parent) doit également être pris dans le premier parent, selon la première règle. Le 2 est au dessus du 4, donc celui-ci doit être pris dans le premier parent. Le 4 est au dessus du 1, ce qui termine le premier cycle. Ensuite nous devons prendre l'élément suivant dans le second parent, c'est le 7 en troisième position. Le 7 est sous le 3, qui est sous le 8, etc. En continuant ainsi, nous obtenons la progéniture suivante :

Enfant1 =	1	2	7	4	5	8	10	3	6	9
Enfant2 =	2	4	3	1	5	6	7	8	9	10

L'avantage de ces trois dernières méthodes est qu'elles ne créent pas de doublons dans les *matching* et les chromosomes ainsi formés n'ont pas une *fitness* nulle.

En plus des méthodes déjà existantes, nous avons créé un *crossover* circulaire qui n'accepte pas les doublons et de plus, conserve le plus de liens possibles.

Crossover circulaire

Il faut imaginer un cercle, sur lequel on place les femmes par ordre croissant et ensuite les hommes de même et on les relie suivant le *matching* obtenu. On prend deux points de coupure au hasard sur le cercle, en faisant attention que le premier soit dans l'ensemble des hommes et le second dans l'ensemble des femmes. Ensuite, on compte le nombre de liens entre les hommes et les femmes qui sont situés entre les deux points de coupure. Si le nombre de liens est suffisamment élevé, on décide d'effectuer le *crossover* et d'échanger les parties coupées. Si lors de l'échange, certains liens sont coupés, les femmes et les hommes correspondants à ces liens se retrouvent célibataires. Voyons cela sur un exemple.

Exemple 4.4.5.

Parent1 =

8	*9*	*5*	*7*	*3*	*1*	*4*	*2*	*10*	*6*

Ce qui se traduit en chromosome circulaire par :

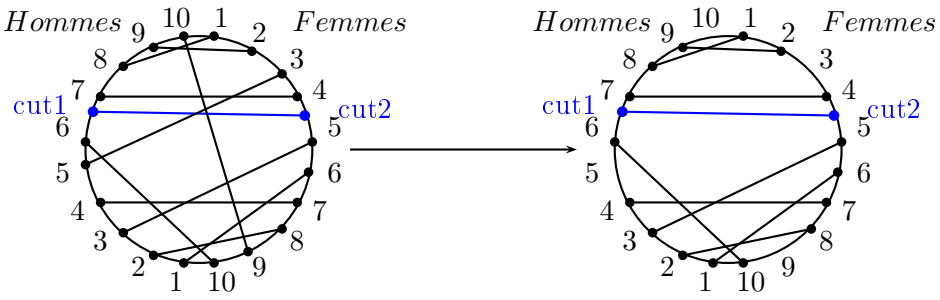

FIGURE 4.1 – Parent 1 : chromosome circulaire

Les liens qui sont coupés dans le parent 1 sont les suivants : entre la femme 9 et l'homme 10 ainsi que entre la femme 3 et l'homme 5. Il va donc rester le chromosome de droite.

Parent2 =

9	*4*	*7*	*8*	*1*	*3*	*6*	*2*	*5*	*10*

Ce qui se traduit en chromosome circulaire par :

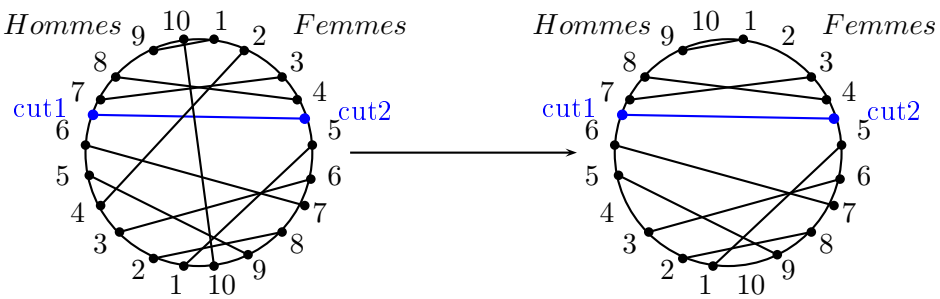

FIGURE 4.2 – Parent 2 : chromosome circulaire

Les liens qui sont coupés dans le parent 2 sont les suivants : entre la femme 2 et l'homme 4 ainsi que entre la femme 10 et l'homme 10. Il va donc rester le chromosome de droite.

Comme le nombre de liens conservés est 6, ce qui est assez élevé, par rapport à un total maximum de 8 liens possibles dans notre configuration (au dessus des points de coupure), on décide de croiser les deux parents. Il reste à échanger les parties coupées, ce qui nous donne :

Enfant1

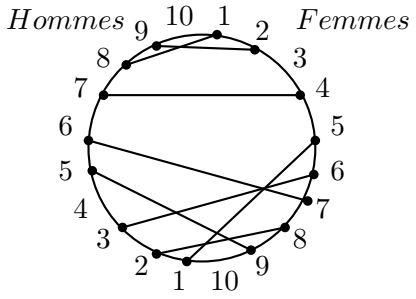

FIGURE 4.3 – Enfant 1 : chromosome circulaire

Les femmes 3 et 8 sont célibataires dans ce nouveau matching. Nous avons donc :

Enfant1 =

8	*9*	*-3*	*7*	*1*	*3*	*6*	*2*	*9*	*-10*

Enfant2

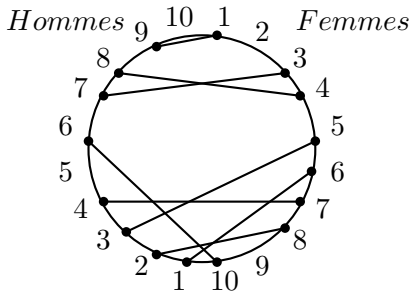

FIGURE 4.4 – Enfant 2 : chromosome circulaire

Les femmes 2 et 9 sont célibataires dans ce nouveau matching mais dans l'ensemble, beaucoup de liens sont conservés. Nous avons donc :

Enfant2 =

9	*-2*	*7*	*8*	*3*	*1*	*4*	*2*	*-9*	*6*

Après le croisement de la population, vient la mutation.

4.5 Mutation

Finalement, pour essayer de trouver de meilleurs individus, il nous reste l'opération mutation. Elle s'effectue sur la population de départ afin de ne pas abîmer les individus créés à partir du *crossover*. La probabilité de mutation de départ est fixée à $p_{mutation}$ et sera modifiée au cours de l'algorithme (voir la Section 4.6).

Nous ne pouvons pas appliquer directement la mutation décrite dans la Section 3.7 car elle ne fonctionne que pour une représentation binaire. Nous avons développé différentes mutations adaptées à notre problème.

Mutation classique

Pour muter le gène choisi, on tire aléatoirement un individu dans la liste des préférences de la femme (locus du gène choisi pour la mutation). On vérifie ensuite si la femme est acceptable pour le nouvel homme. Si c'est le cas, cet individu lui est associé, sinon, la femme se retrouve célibataire.

Lorsque le nombre d'individus devient trop grand, les *matching* sont de moins en moins souvent stables et nous introduisons une fonction qui rend les *matching* stables.

Mutation stabilisante

Afin de pallier à ce problème, nous avons décidé de rajouter à la mutation une fonction qui stabilise certains chromosomes de la population. D'abord, on mute la population normalement, en utilisant la procédure décrite ci-avant et ensuite, on stabilise 10% de la population. La stabilisation se fait en échangeant les couples qui bloquent le *matching*, selon la définition donnée à la Définition 2.1.2.

Cette mutation fonctionne bien mais elle peut être améliorée.

Mutation conditionnée

Cette mutation est identique à la mutation stabilisante, si ce n'est que, dans la phase de mutation classique, au lieu de choisir aléatoirement le nouvel individu dans toute la liste des préférences de la femme, il est choisi parmi les individus qui ont un plus haut *ranking* que l'individu de départ. On aura que la femme préfère son partenaire muté à celui qu'elle avait avant. Pour plus de stabilité, nous ajoutons la fonction stabilisante.

Mutation simple

Comme notre problème veut que chaque homme ne soit couplé qu'à une seule femme, nous pouvons adapter la mutation afin qu'elle respecte cette contrainte. En effet, supposons que nous avons un *matching* qui respecte cette contrainte, si nous utilisons la mutation décrite à la Section 4.5, il se peut qu'un homme soit couplé avec deux femmes. Dès lors nous perdons notre belle propriété et la *fitness* du nouveau *matching* sera égale à 0. Pour éviter cela, nous échangeons les valeurs de deux gènes, ainsi chaque homme reste marié à une seule femme.

Les chromosomes de la population de départ ont été modifiés grâce au *crossover* et à la mutation. A présent il s'agit de choisir les meilleurs individus parmi ces deux nouvelles population et celle de départ. On applique simplement l'opérateur de sélection et ainsi, les chromosomes sélectionnés forment la nouvelle population de départ pour une nouvelle étape de l'algorithme génétique.

Nous pouvons également calculer la diversité de la population formée.

4.6 Diversité

La diversité calcule la proximité des chromosomes, les uns par rapport aux autres. Elle se calcule à la fois sur la population mais également sur la *fitness*. la première composante est la distance moyenne entre tous les chromosomes et la seconde est la variance des *fitness*.
La diversité est utile car elle permet de savoir si l'algorithme ne converge pas trop vite vers un seul individu (minimum local). Pour éviter cela, lorsque la diversité est trop faible, on augmente le taux de mutation afin que la population couvre un large éventail de *matching*.

A la page suivante, vous trouverez un diagramme des différents blocs qui peuvent être utilisés pour construire l'algorithme génétique.

4.7 En pratique

Pour plus de facilité, nous avons choisi de coder tous les algorithmes en Matlab. En plus de l'algorithme génétique, composé de tous les blocs ci-dessus, nous avons également codé l'algorithme de Gale & Shapley afin de pouvoir les comparer. Différents tests sur les paramètres ont été effectués dont vous trouverez les résultats dans la section suivante.

FIGURE 4.5 – Diagramme des blocs de l'algorithme génétique

4.8 Résultats

Nous avons effectué différents tests afin de mieux caractériser les algorithmes génétiques. Attention, certains résultats peuvent différer, du fait que la population de départ est générée aléatoirement et que toutes les fonctions utilisées sont aussi basées sur des processus aléatoires.

Mutation

Le premier test sert à déterminer la meilleure technique de mutation, quelle que soit la méthode de *crossover*, en fonction du nombre de participants. La méthode d'évaluation de la *fitness* utilisée est la méthode qui considère les préférences des hommes et des femmes et la stabilité du *matching*. Les paramètres utilisés sont les suivants :

- Le nombre d'hommes est égal au nombre de femmes, qui est égal à N et détermine la taille du groupe (2N) ;
- La longueur des préférences des femmes et des hommes est égale à N ;
- $p_{crossover}$=0.4 ;
- $p_{mutation}$=0.1 ;
- Nombre d'étapes (générations) de l'algorithme=100 ;
- Taille de la population de départ=50 chromosomes si le nombre de femmes est inférieur à 10, et 400 sinon.

Voici un graphe représentatif pour $N = 10$.

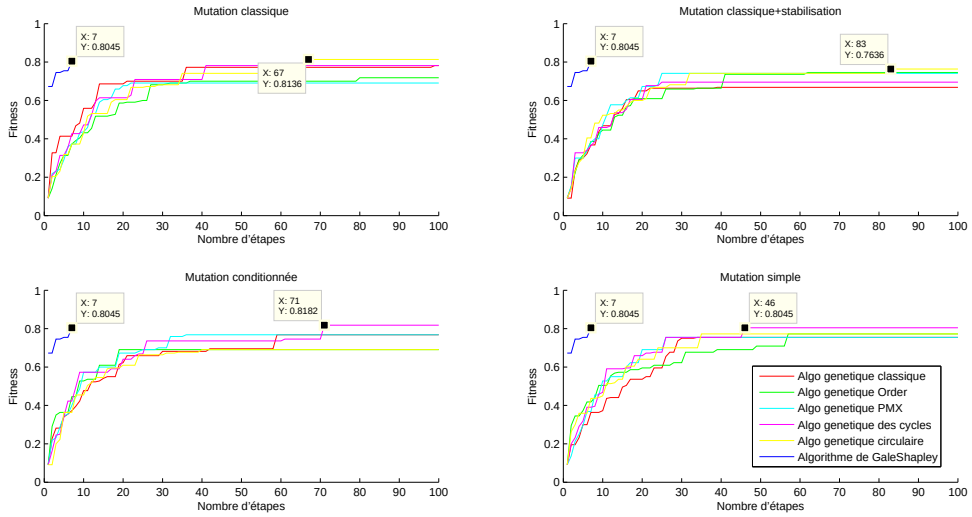

FIGURE 4.6 – Test de la mutation, 10 individus : *fitness* en fonction du nombre d'étapes. En haut à gauche, la mutation classique, à droite la mutation stabilisante, en bas à gauche la mutation conditionnée et à droite la mutation simple, avec en rouge le GA classique, en vert le GA *order*, en turquoise le GA PMX, en magenta le GA des cycles, en jaune le GA circulaire et en bleu le DAA

Les deux seules mutations qui permettent de dépasser l'utilité obtenue par le DAA, sont la mutation classique et la mutation conditionnée. Cependant, les autres mutations ont été crées pour pallier à des problèmes qui surgissent lorsque le nombre d'individus augmente. Dès lors regardons ce qui se passe si le nombre de femmes est égal à 20.

Voici un graphe représentatif pour $N = 20$.

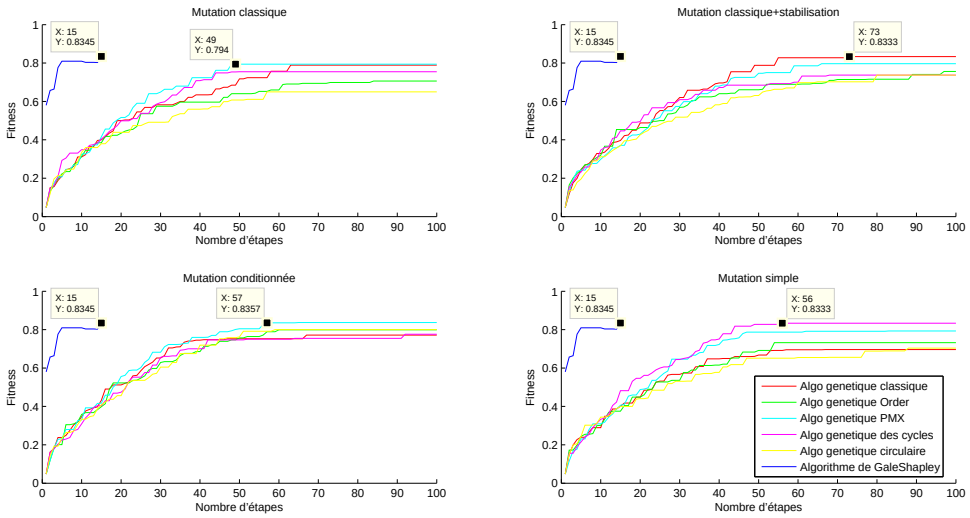

FIGURE 4.7 – Test de la mutation, 20 individus : *fitness* en fonction du nombre de générations. En haut à gauche, la mutation classique, à droite la mutation stabilisante, en bas à gauche la mutation conditionnée et à droite la mutation simple, avec en rouge le GA classique, en vert le GA *order*, en turquoise le GA PMX, en magenta le GA des cycles, en jaune le GA circulaire et en bleu le DAA

Lorsque le nombre d'individus augmente, toutes les méthodes n'arrivent plus à trouver une utilité aussi élevée que l'algorithme de Gale & Shapley. Cependant, avec la mutation conditionnée, le *crossover* PMX trouve une meilleure *fitness* que le DAA et pour l'ensemble des méthodes de *crossover*, c'est également avec cette méthode de mutation que l'on obtient les meilleurs résultats. De plus, le nombre d'étapes pour trouver la meilleure *fitness* est raisonnable (57 itérations). En conclusion, la mutation conditionnée donne de meilleurs résultats, nous l'utiliserons donc pour la suite des tests.

Stabilité

Le second test effectué est un test sur la fonction d'évaluation de la *fitness*. Ce test compare l'évaluation de *fitness* qui considère les préférences des hommes et des femmes ainsi que la stabilité des *matching* et celle qui ne considère que les préférences des femmes et des hommes (pas la stabilité). Le premier algorithme génétique utilise la mutation conditionnée et pénalise les *matching* qui ne sont pas stables et le second utilise la mutation simple et ne pénalise

aucun *matching*. Les paramètres utilisés sont les suivants :

- Le nombre d'hommes est égal au nombre de femmes, qui est égal à N et détermine la taille du groupe (2N) ;
- La longueur des préférences des femmes et des hommes est égale à N ;
- $p_{crossover}$=0.6 ;
- $p_{mutation}$=0.1 ;
- Nombre d'étapes de l'algorithme = 100 ;
- Taille de la population de départ = 100 chromosomes si le nombre de femmes est inférieur à 10, et 400 sinon.

Remarque : dans l'algorithme de Gale & Shapley, la fonction d'évaluation n'a aucune sorte d'importance car nous savons que l'algorithme donne un *matching* stable.

Les résultats suivants sont la moyenne de 50 exécutions, pour N allant de 5 à 20.

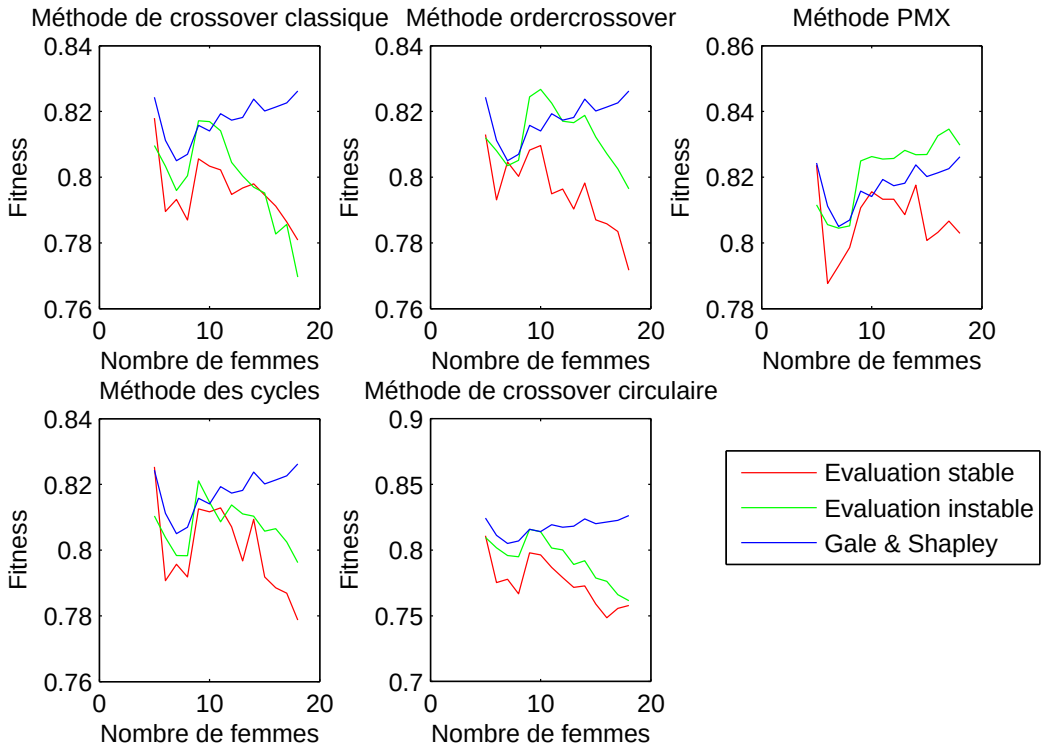

FIGURE 4.8 – *Fitness* versus stabilité : longueur des préférences égale à N. En haut à droite, le Xover classique, au milieu l'*ordercrossover*, à droite le *Xover* PMX, en bas à droite le *Xover* des cycles et au milieu le *Xover* circulaire avec en rouge la *fitness* obtenue par l'évaluation stable, en vert par l'évaluation instable et en bleu la *fitness* obtenue par le SDAA.

Nous constatons que en général la *fitness* instable est supérieure à la *fitness* stable et de plus, la *fitness* instable est supérieure à celle obtenue par le DAA, lors de l'utilisation du *crossover*

PMX. Par ailleurs, plus il y a de participants et plus cette méthode est performante. Il faut faire attention que toutes les méthodes de *crossover* ne le sont pas.

Si maintenant, nous diminuons la longueur des préférences, que se passe-t-il au niveau de la stabilité ?
La longueur des préférences est fixée à 10, nous commençons dès lors les tests à partir de 10 femmes jusque 20. Voici l'utilité moyenne obtenue sur 50 exécutions.

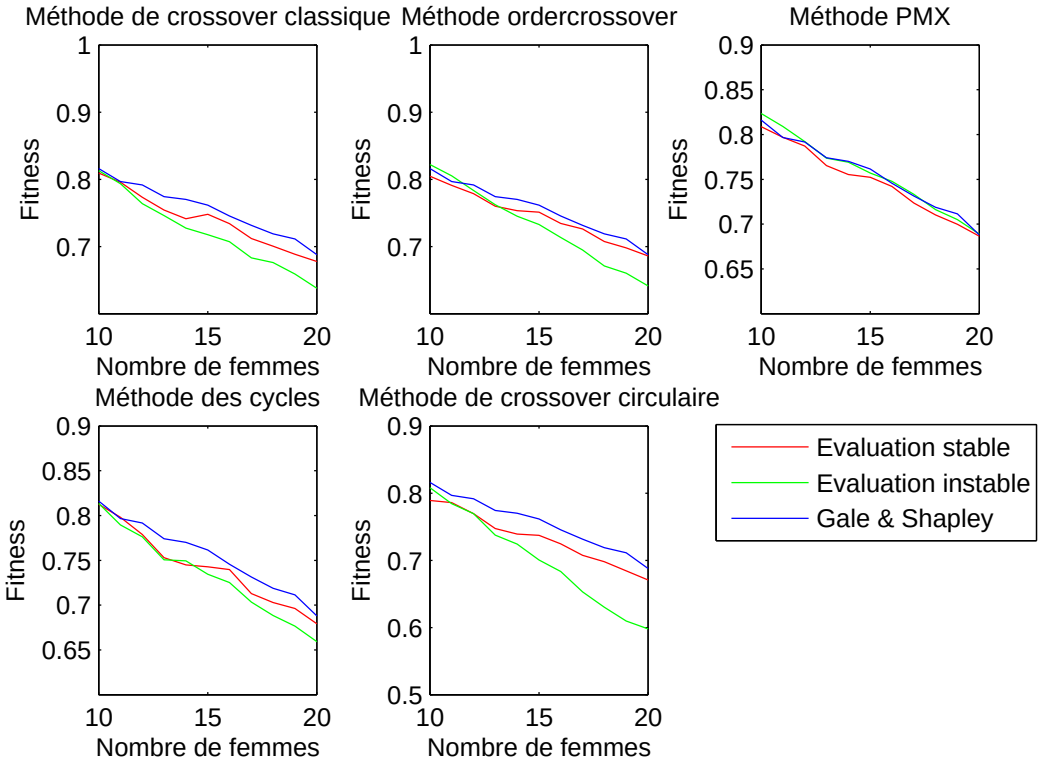

FIGURE 4.9 – *Fitness* versus stabilité : longueur des préférences = 10. En haut à droite, le Xover classique, au milieu l'*ordercrossover*, à droite le *Xover* PMX, en bas à droite le *Xover* des cycles et au milieu le *Xover* circulaire avec en rouge la *fitness* obtenue par l'évaluation stable, en vert par l'évaluation instable et en bleu la *fitness* obtenue par le SDAA.

Nous remarquons que l'algorithme génétique muni du *crossover* PMX obtient de très bons *matching* dont la *fitness* est parfois plus élevée que celle du *matching* obtenu par le DAA et que la *fitness* instable est plus élevée que celle stable. La question se pose à présent de quelle approche préférer. Vaut-il mieux un *matching* stable mais avec une utilité moindre ou un *matching* instable qui apporte plus de satisfaction générale ? Selon la théorie utilitariste, il faudrait choisir la seconde option mais nous laisserons cette question ouverte à la réflexion du lecteur.

Temps d'exécution

Ce test considère le temps d'exécution des algorithmes génétiques, un pour chaque méthode de *crossover* différente et le temps de l'algorithme de Gale & Shapley. Les fonctions utilisées sont les suivantes : *fitness* qui considère les préférences des hommes et des femmes ainsi que de la stabilité des *matching* et mutation conditionnée. Les paramètres sont les suivants :

- Le nombre d'hommes est égal au nombre de femmes, qui est égal à N ;
- La longueur des préférences des femmes est fixée à 10 ;
- $p_{crossover}$=0.45 ;
- $p_{mutation}$=0.1 ;
- Nombre d'étapes de l'algorithme (générations) =100 ;
- Taille de la population de départ=500 si le nombre de femmes est plus petit que 20, 600 s'il est plus petit que 30, 700 s'il est plus petit que 40 et 800 s'il est plus petit que 50.

Comme les algorithmes génétiques ne donnent pas de résultat déterministe, nous avons effectué 50 simulations à chaque variation du nombre de femmes (de 10 à 50). Les résultats suivants en sont la moyenne.

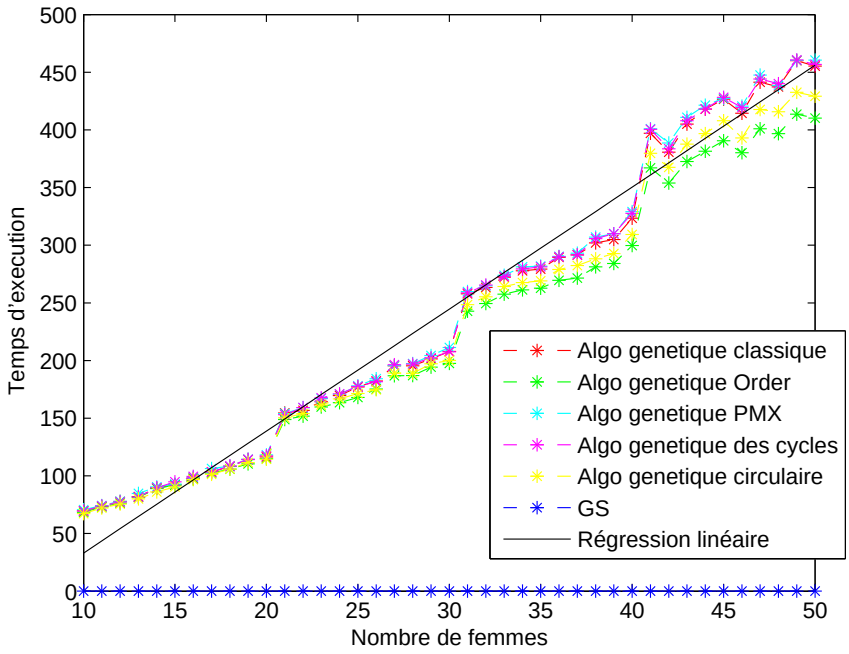

FIGURE 4.10 – Temps moyen d'exécution des algorithmes génétiques et de l'algorithme de Gale & Shapley, en rouge le GA classique (var=259.58), en vert le GA *order* (var=179.05), en turquoise le GA PMX (var=264.84), en magenta le GA des cycles (var=271.62), en jaune le GA circulaire (var=201.06) et en bleu le DAA (var=$4e-005$). Entre parenthèse, est reportée la variance moyenne pour chaque algorithme.

On constate que les algorithmes génétiques sont beaucoup plus lents que l'algorithme de Gale & Shapley. On sait pourtant que théoriquement le DAA a une complexité de $O(N^2)$

mais il ne traite qu'une seule solution alors que l'algorithme génétique en traite beaucoup plus (taille de la population de départ). On ne peut dès lors pas vraiment comparer ces deux temps d'exécution. Il semble cependant que le temps d'exécution des algorithmes génétiques augmente linéairement par rapport à la taille du problème, mais l'échantillon est trop petit pour être représentatif. Cette question pourrait être analysée plus en profondeur cependant, ce n'est pas le but de ce travail.

Dès lors, qu'ont apporté les algorithmes génétiques ?

Efficience des algorithmes génétiques

Pour ce dernier test, nous avons utilisé la mutation conditionnée et la fonction d'évaluation de *fitness* qui tient compte de la stabilité. Les paramètres sont identiques à ceux utilisé pour le test du temps d'exécution. Ce graphe correspond à la moyenne des meilleures *fitness* obtenues sur 50 simulations.

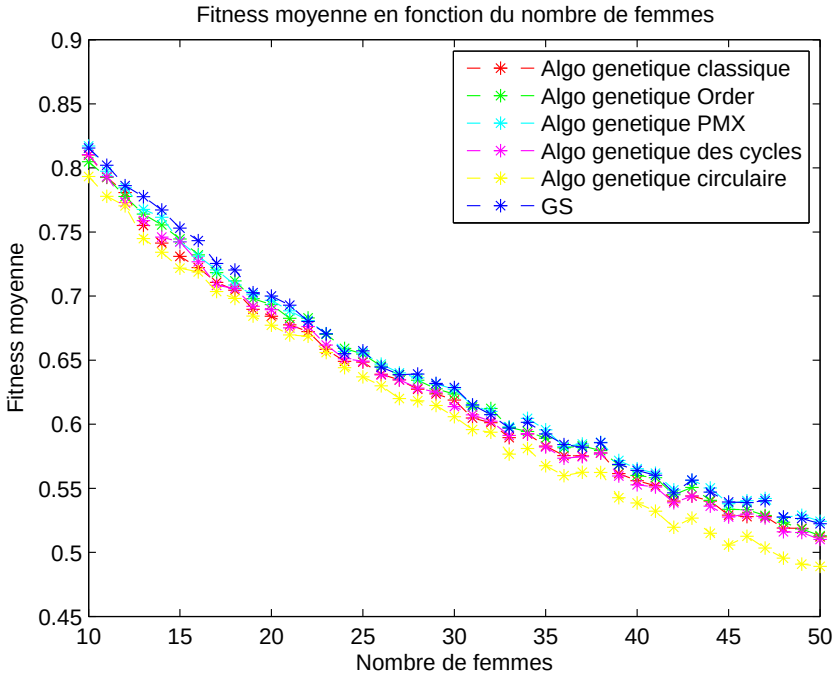

FIGURE 4.11 – *Fitness* moyenne des algorithmes génétiques et de l'algorithme de Gale & Shapley, en rouge le GA classique (var=0.0011), en vert le GA *order* (var=0.0010), en turquoise le GA PMX (var=0.0010), en magenta le GA des cycles (var=0.0011), en jaune le GA (var=0.0012) circulaire et en bleu le DAA (var=9e-004). Entre parenthèse, est reportée la variance moyenne pour chaque algorithme.

Voici un zoom de 10 à 30 femmes :

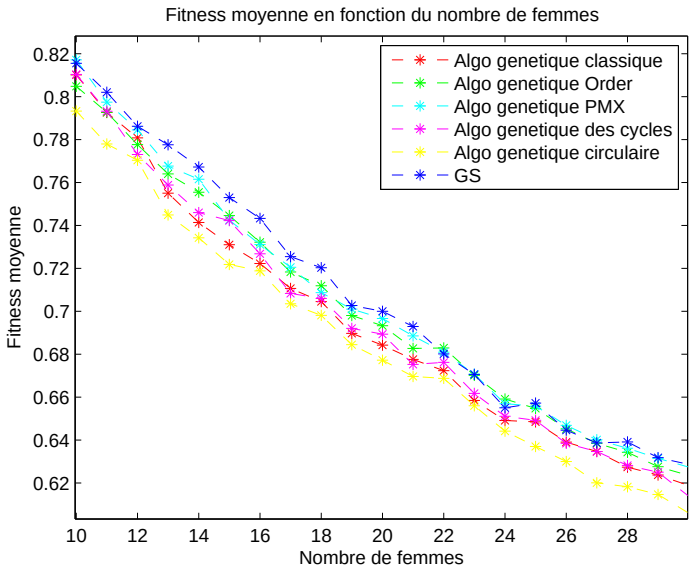

FIGURE 4.12 – *Fitness* moyenne des algorithmes génétiques et de l'algorithme de Gale & Shapley, zoom de 10 à 30 femmes, en rouge le GA classique, en vert le GA *order*, en turquoise le GA PMX, en magenta le GA des cycles, en jaune le GA circulaire et en bleu le DAA.

Voici un zoom de 30 à 50 femmes.

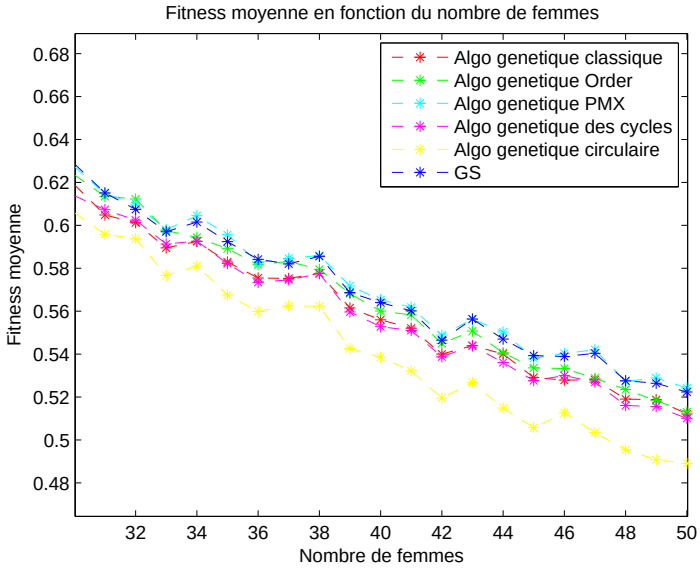

FIGURE 4.13 – *Fitness* moyenne des algorithmes génétiques et de l'algorithme de Gale & Shapley, zoom de 30 à 50 femmes, en rouge le GA classique, en vert le GA *order*, en turquoise le GA PMX, en magenta le GA des cycles, en jaune le GA circulaire et en bleu le DAA.

Nous pouvons constater que l'algorithme DAA donne une meilleure *fitness* que les algorithmes génétiques entre 10 et 30 femmes. Cependant, de 30 à 50 femmes, l'algorithme génétique muni du *crossover* PMX (nous l'appellerons AGPMX) donne une *fitness* légèrement supérieure à celle du DAA. Même si cette amélioration est faible, lorsque le nombre de préférences diminue par rapport au nombre de participants totaux, le AGPMX donne une meilleure *fitness* que le DAA et de plus le *matching* obtenu est stable puisque nous utilisons l'évaluation de la *fitness* qui tient compte de la stabilité du *matching*. Nous pouvons obtenir une telle amélioration car nous considérons à la fois les préférences des femmes et celles des hommes alors que le DAA ne considère que les préférences des femmes (lorsque les hommes proposent et vice versa). Il est vrai que le DAA donne le meilleur *matching* pour les femmes mais c'est également le pire *matching* pour les hommes. Grâce à notre approche, nous offrons un *matching* mitigé qui ne favorise aucune partie puisque les algorithmes génétiques sont un procédé aléatoire.
Pour mieux comprendre ce qu'il se passe lors d'une simulation, voici le résultat en terme de choix obtenu. La figure suivante illustre, pour $N = 45$ et longueur des préférences fixée à 10, le pourcentage d'individus (hommes et femmes) qui obtiennent au moins leur premier choix, second, troisième, etc.

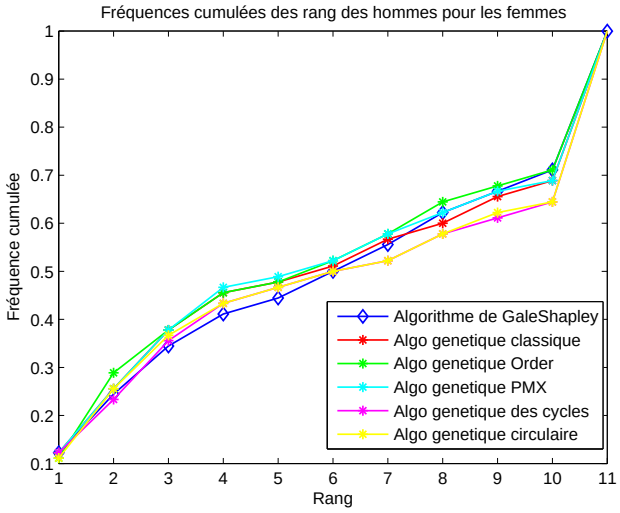

FIGURE 4.14 – Fréquence cumulée du rang obtenu dans le *matching*, $N = 45$, longueur des préférence =10, en rouge le GA classique, en vert le GA *order*, en turquoise le GA PMX, en magenta le GA des cycles, en jaune le GA circulaire et en bleu le DAA.

On observe que avec le GA PMX et *order*, il y a plus de gens qui reçoivent au moins leur premier choix, second etc jusqu'au dixième choix (dernier sur la liste des préférences) qu'avec le DAA. Les gens qui ne sont pas associés avec quelqu'un de leur liste, sont considérés dans la onzième préférence. Nous avons donc une meilleure répartition des hommes et des femmes ce qui se traduit par un gain de rang dans la liste des préférences et une *fitness* plus élevée (cfr. Figure 4.13).

Remarque :

Après avoir terminé notre modélisation, nos codes et les tests décrits plus haut, nous avons trouvé dans la littérature un article qui se rapproche assez bien de ce qu'on a fait dans ce chapitre. En effet, dans [AC99], Aldershof et al., appliquent les algorithmes génétiques au problème du *many-to-one-matching*. Dans un premier temps, ils réduisent le *many-to-one* à un problème de mariage et cherchent à énumérer les *matching* stables pour ce problème. La seconde partie de l'article concerne le problème des couples de médecins qui cherchent un stage dans un hôpital[2] mais étant donné que nous n'avons pas travaillé sur ce sujet, nous ne parlerons que du problème du mariage. Décrivons maintenant les différences entre notre approche et celle d'Aldershof et al. Premièrement, ils définissent leur *fitness* en terme de contraintes d'égalité et d'inégalité. Ce point de vue est intéressant car il évite la transformation arbitraire des préférences ordinales en préférences cardinales. Cependant, cette *fitness* ne permet pas de comparer entre eux les *matching* stables obtenus car ceux-ci ont toujours la même *fitness* (en effet, dans chaque *matching* stable, ce sont les mêmes personnes qui ont un partenaire), alors que notre *fitness* sert de critère de comparaison entre les différents algorithmes. De plus, ils n'imposent pas que leurs *matching* soient stables, ce qui implique qu'ils n'obtiennent pas nécessairement des *matching* stables. Les *matchings* de la population initiale, sont construits de manière aléatoire, alors que nous ne prenons que des couples acceptables ou le *matching* dans lequel tout le monde est célibataire. Pour ce qui est du *crossover*, ils utilisent le *cyclic crossover*, développé pour le TPS et ne mentionnent pas les autres *crossover* du TPS. Par ailleurs, nous comparons les différents *crossover* et il s'avère que dans notre cas, le PMX donne de meilleurs résultats. En plus de la reproduction, ils introduisent 2 nouveaux opérateurs : *restart* et inversion, qui en quelque sorte améliorent leur *crossover*. Ces opérateurs sont intéressants car ils rendent le *cyclic crossover* plus performant. Cependant, l'algorithme devient plus coûteux et laisser faire l'évolution ainsi qu'utiliser le *crossover* PMX est suffisant. Pour sélectionner les chromosomes qui se reproduisent, ils utilisent une stratégie différente mais pas assez explicitée dans l'article. Leur mutation est identique à la nôtre (voir Section 4.5)) et pour la sélection, ils remplacent notre roulette *remainder stochastic* par une sélection tournoi (expliquée plus en détail dans [AC99]). Enfin, comme résultat, ils obtiennent 3 *matching* stables différents, l'optimal pour les femmes, l'optimal pour les hommes et un troisième entre les deux précédents. Nous trouvons aussi un *matching* qui n'est ni optimal pour les hommes, ni pour les femmes et dont la *fitness* dépasse celle du *matching* optimal pour les femmes. Nous introduisons plus d'opérateurs et les testons sur différents critères, longueur des préférences, temps d'exécution, etc. Pour conclure leur papier, Aldershof et al. parlent d'utiliser les algorithmes génétiques dans d'autres problèmes de *matching* mais à notre connaissance, aucun autre article n'a été publié, ayant rapport avec ce que nous faisons pas la suite.

Faire le point

Dans ce chapitre nous avons créé des algorithmes génétiques afin de les comparer au DAA de Gale & Shapley. Les résultats obtenus sont très concluants : les*matching* obtenus ont une *fitness* plus élevée que celui produit par le DAA, surtout dans le cas où le nombre de préférences des individus est inférieur au nombre d'individus. En d'autres termes, certains individus sont couplés avec un partenaire qu'ils préfèrent à celui du *matching* du DAA. De plus, le temps d'exécution, même s'il est beaucoup plus élevé que celui du DAA, reste raisonnable et donc ces algorithmes sont tout à fait réalisables. Le temps d'exécution est

2. Plus de détails dans [AC99].

compensé par l'obtention de meilleurs résultats.

Si le *matching* que nous obtenons n'est plus optimal pour l'un des deux groupes, il n'est plus non plus le pire *matching* possible pour l'autre groupe. Nous offrons un *matching* qui considère à la fois les préférences des hommes et des femmes.

Nous n'avons pas analysé la question de la *strategy-proofness* mais étant donné que le DAA n'est déjà pas *strategy-proof*, cette question n'est pas très importante. Cependant, comme les algorithmes génétiques ne sont pas déterministes, nous pouvons dire qu'ils sont moins manipulables. Il est difficile de savoir comment modifier ses préférences pour obtenir un meilleur résultat qu'avec ses vraies préférences.

Nous mettons encore une fois l'accent sur le fait que la *fitness* utilisée est arbitraire et donc pas tout à fait représentative. Il suffirait de recommencer les tests avec une autre *fitness* (différente représentation des préférences des individus) pour savoir si nos algorithmes sont vraiment efficients. Mais ce n'est pas le but de ce travail.

En conclusion, nous sommes sur la bonne voie pour trouver des solutions au problème belge. Ce que nous faisons d'ailleurs, dans le chapitre suivant.

Chapitre 5

Le cas Belge

"The study of the discrete two-sided matching models with non-necessarily strict preferences and the search for algorithms to produce the Pareto-stable matching is a new and interesting line of investigation."
Sotomayor and Ozak (Two-sided matching models)

Pour l'année académique 2011-2012, 42 709 élèves convoitaient 62 778 places dans différentes écoles de la communauté française mais le premier septembre 2011, il restait encore 163 élèves sans école. Le problème est principalement situé dans la région de Bruxelles, avec au premier septembre 39% des écoles qui étaient complètes, et donc dans l'impossibilité de satisfaire les demandes excédentaires [1].
Il est évident que face à de telles situations, les parents paniquent et s'interrogent sur la fiabilité du système des inscriptions scolaires. De plus, il est difficile pour les écoles d'organiser leurs classes de premières années si de nouveaux élèves arrivent au début du mois de septembre. Enfin, des parents se plaignent de certaines écoles qui les décourageraient à inscrire leur enfant chez elles. Les objectifs de mixité sociale ne sont pas vraiment atteints et certaines écoles ne donnent pas les chiffres corrects quant au nombre de places réelles disponibles en première année.
Dans ce chapitre, nous allons analyser en détail le décret d'inscriptions belge afin comprendre le système, avec pour but d'essayer de l'améliorer.

5.1 Le décret

Le décret "missions", que vous trouverez plus en détail dans [Decret11], vise à réguler les inscriptions des élèves au sein des établissements scolaires dans le premier degré de l'enseignement secondaire et à favoriser la mixité sociale.

A qui s'applique le décret ?

C'est le chef de l'établissement scolaire qui doit appliquer le décret "missions" dans son établissement, en fonction des élèves qui s'y inscrivent. Ce n'est que lorsqu'une école reçoit plus de demandes d'inscription qu'elle n'a de places disponibles, que les demandes excédentaires sont envoyées à la Commission Interréseaux des Inscriptions (CIRI). Dès lors, celle-ci répartit les élèves dans les écoles incomplètes. Il n'y a donc aucune vraie centralisation de

1. Données tirées du site européen, sur le *matching*, [BCK11]

l'information, ce qui cause une mauvaise gestion des inscriptions.

Tous les élèves (et leurs parents) qui s'inscrivent en première année dans l'enseignement secondaire (de la communauté française et Bruxelles) sont également concernés par ce décret.

Comment fonctionne-t-il ?

Toute demande d'inscription en première année de l'enseignement secondaire est faite grâce à un formulaire unique d'inscription (FUI).
Celui-ci est tout d'abord complété par l'administration, pour chaque élève inscrit en $6^{ème}$ primaire. Il comporte le nom, prénom et date de naissance de l'enfant, son domicile ainsi qu'un code indiquant si l'élève est ou non considéré comme venant d'une implantation scolaire moins favorisée (élève ISEF) et enfin, l'indice socio-économique du quartier d'origine de l'élève. Ces formulaires sont ensuite envoyés aux parents de l'élève, qui doivent complèter le formulaire avec le nom de l'établissement secondaire qui correspond le mieux à leurs préférences. En plus de cette première école, ils en ajoutent maximum 9 autres dans l'ordre décroissant de leurs préférences. Cette liste de 10 écoles forme les préférences des élèves et reste secrète pour l'ensemble des autres élèves. Cependant, il est parfois difficile pour les parents de nommer 10 écoles car ils ne les connaissent pas bien et souvent, dans ce genre de cas, la liste est incomplète [2]. Normalement, l'élève ne peut déposer qu'un seul formulaire FUI dans son établissement scolaire préféré mais on assiste parfois à des doubles dépositions, ce qui entraine l'inscription d'un élève dans différents établissements et ensuite l'annulation par le CIRI de tous les FUI déposés.

Du classement des élèves

Les écoles, lorsqu'elles ont les FUI, classent les élèves en fonction de différents critères.

Quota ISEF

Tout d'abord, les établissements doivent attribuer au moins 20.4% de leurs places aux élèves ISEF (quota minimum). Bien évidemment, ceci, seulement si suffisamment d'élèves ISEF ont introduit une demande d'inscription. Si le nombre de demandes excède les 20.4%, les élèves sont acceptés dans l'ordre de leur indice composite (voir sous- section suivante)

Elèves prioritaires

Ensuite, parmi les places restantes, le directeur attribue une place, aux élèves prioritaires et puis au reste des élèves. Sont considérés comme prioritaires, les élèves :

1. Dont un frère ou une soeur (ou tout autre mineur résidant sous le même toit) fréquente déjà l'établissement scolaire ;
2. Qui sont issus :
   - D'un home ou d'une famille d'accueil ;
   - D'un internat pour les enfants dont les parents n'ont pas de résidence fixe ;

2. Nous ne disposons pas de données pour le cas belge mais à New York, lors des inscriptions, 72 à 80% des étudiants listent moins de 12 écoles (nombre maximal d'écoles admises), cfr. [APR09].

   - D'un centre d'accueil.
3. Sortant de l'enseignement spécialisé ou éprouvant des besoins spécifiques fondés sur un handicap avéré ;
4. Fréquentant un enseignement primaire dépendant du même pouvoir organisateur que l'école secondaire, ou adossé à celle-ci ;
5. Dont un parent travaille dans l'établissement secondaire.

Ces conditions sont plus détaillées dans [Decret11].

Indice composite

Finalement, les places restantes sont attribuées aux élèves en fonction de leur indice composite. Au départ l'indice composite est égal à 1 ; il est ensuite multiplié par les facteurs suivants :

1. La proximité de l'école *primaire* au domicile de l'élève, la pondération est dégressive, en fonction de la première plus proche, à la cinquième plus proche. Respectivement, les facteurs multiplicatifs sont 2, 1.81, 1.61, 1.41 et 1.21 de la première à la cinquième école la plus proche. Attention, par distance, il faut entendre la distance à vol d'oiseau ;
2. La proximité de l'école *secondaire* au domicile de l'élève, la pondération est dégressive, en fonction de la première plus proche, à la cinquième plus proche. Respectivement, les facteurs multiplicatifs sont 1.98, 1.79, 1.59, 1.39 et 1.19 de la première à la cinquième école la plus proche ;
3. L'établissement secondaire choisi se situe dans un rayon de 4 km de l'école primaire, si oui, le facteur est 1.54 et sinon, 1 ;
4. Il existe un partenariat entre l'école secondaire et l'école primaire, visant à favoriser la transition du primaire au secondaire. Cependant, ce critère n'est considéré que si au moins 3 écoles primaires sont concernées et au moins l'une d'elles est considérée comme moins favorisée. Ce critère vaut 1.51 s'il est rencontré et sinon 1 ;
5. L'école secondaire offre la possibilité de poursuivre en immersion dans la même langue que celle de l'école primaire (au moins à partir de la $3^{ème}$ primaire). Ce critère vaut 1.18 et sinon 1.

Ces facteurs multiplicatifs permettent de classer les élèves et de leur donner un ordre d'acceptation. Lorsque deux élèves ont le même indice composite, ils sont classés par ordre croissant de l'indice socio-économique de leur quartier d'origine. L'indice composite sert de *tie-breaking* : c'est une façon de départager les ex-aequo.

Si malheureusement, les informations nécessaires au calcul de l'indice composite sont insuffisantes, l'élève se voit attribuer la moyenne des indices composites des autres élèves qui ont fait une demande d'inscription dans le même établissement.

Nous constatons que le décret belge mélange à la fois des contraintes de mixité sociale non strictes (au moins 20.4 % d'élèves ISEF Section 2.4) et un classement des autres élèves selon différents critères d'acceptation.

Pratiquement, il y a deux phases d'enregistrement [3]. A la fin de la première, les écoles qui sont incomplètes inscrivent de manière permanente tous leurs élèves, les autres en inscrivent 80%, en respectant les contraintes citées ci-dessus. Les élèves en trop et les élèves s'inscrivant durant la seconde phase d'enregistrement sont assignés par le CIRI, selon les critères décrits ci-dessus.

De la part des parents, cela ne serait pas du tout stratégique de ne pas s'inscrire durant la première phase d'inscription car ils perdraient une chance d'inscrire leur enfant dans l'école de leur premier choix. On fera donc l'hypothèse que tous les parents inscrivent leurs enfants durant la première phase d'inscription.

Le premier point faible de ce décret est qu'il n'y a aucune centralisation de l'information, ce qui rend les procédures administratives complexes et fastidieuses, surtout pour les parents et les directeurs. Une centralisation de l'information serait peut-être dure à accepter de la part des écoles car elle entraine une perte d'autonomie de celles-ci. Mais cela simplifierait la procédure et il n'y aurait plus le problème des élèves sans école en septembre. Par la suite, nous faisons l'hypothèse d'une centralisation des demandes d'inscription et nous proposons un algorithme qui assigne les élèves en tenant compte des critères du décret.

5.2 Modélisation de la population

Nous ne disposons pas de données réelles mais étant donné qu'il n'y a aucune centralisation de l'information, les données seraient biaisées et donc inutilisables. De plus, le problème de la protection de la vie privée se poserait. Dès lors, nous avons décidé de construire un modèle simplifié de population synthétique.

Tout d'abord, appliquer le modèle à toute la communauté française dépassait le temps et les moyens accordés à ce mémoire et nous nous sommes restreints à l'arrondissement de Namur. De même, des données telles que l'école primaire ou encore si un frère ou une soeur est déjà dans l'école secondaire désirée, sont impossibles à modéliser. Nous n'avons modélisé que ce qui était plausible, à savoir pour chaque élève, une localisation géographique approximative de son domicile et un indice ISEF.

La localisation géographique est estimée à partir d'une carte de l'arrondissement de Namur. Chaque commune, ou quartier de Namur est représenté par un point en son centre, qui est la localisation géographique des élèves y habitant. Pour chaque commune, le nombre d'élèves est une approximation du nombre d'enfants de 12 ans (moyenne d'âge pour rentrer en secondaire) habitant cette commune (chiffres extraits de [VN10]) et le nombre total d'élèves est de 1750.

Dans notre modèle, l'indice composite de l'élève s pour l'école c est noté $I_{s,c}$ et ne tient compte que de la distance du domicile de l'élève à l'école choisie car nous ne disposons pas d'autres informations. $I_{s,c}$ est donc une mesure restreinte par rapport au décret mais néanmoins facilement adaptable si nous disposions de données complètes. Pour le calcul exact de l'indice composite, cfr. Section 5.1.

Chaque étudiant possède également un indice ISEF, représenté par 0 si l'enfant n'est pas

3. Pour plus de détail sur le fonctionnement des phases d'enregistrement, consulter le décret [Decret11]

ISEF et 1 s'il l'est. Cet indice est tiré aléatoirement avec une probabilité de 0.2 pour chaque élève d'être ISEF. En effet, dans le décret il est stipulé que dans chaque école, il doit y avoir 20% d'élèves ISEF. Nous avons supposé qu'il y avait en moyenne 20% d'élèves ISEF sur toute la population d'élèves. Cependant, pour aller plus loin, nous pourrions tester les algorithmes avec différents pourcentages d'élèves ISEF.

Pour ce qui est des écoles, nous avons choisi les 15 écoles proposées par le site de la Fédération Wallonie-Bruxelles pour l'arrondissement de Namur. Leur localisation géographique est détaillée dans [FWB].

Le nombre d'élèves que chaque école peut accueillir en première année n'est pas une donnée publique et nous avons décidé de prendre des petites écoles, des moyennes et des grandes. La capacité des écoles sera adaptée afin de rester cohérent avec le nombre d'élèves utilisés dans les tests effectués a posteriori.

Enfin, nous avons créé les préférences des élèves aléatoirement en prenant, pour chaque élève, 5 écoles au hasard dans les 15 proposées. Il est évident que les préférences sont indépendantes et considérées comme les choix réels des élèves (*strategy-proofness*).

5.3 Algorithme SDAA

Dans ce problème belge, nous sommes confrontés à un nouveau problème : les préférences des écoles ne sont plus strictes. En effet, elles sont basées sur l'indice composite des élèves et ce dernier peut être identique pour plusieurs élèves. Il faut dès lors choisir une règle pour briser les ex-aequo (*tie-breaking rule*) afin de départager les élèves. Dans le décret belge, ils utilisent l'indice socio-économique du quartier de l'élève mais nous ne disposons pas de cette information. Quoi qu'il en soit, cela n'a pas beaucoup d'importance car nous montrerons que lorsqu'on introduit une règle de *tie-breaking*, cela entraine une perte d'efficience des algorithmes (perte d'utilité pour les élèves).

Dans un premier temps, nous avons choisi l'approche *soft bounds* et avons adapté le SDAA-SB (voir Section 2.4) à nos contraintes assez simples. En effet, ici le quota minimal ISEF est de 20%. Le quota maximal ISEF n'étant pas défini, il est posé à 100%. Le quota minimal pour les autres élèves est de 0% et le quota maximal est de 80% par défaut. La seule différence est que les préférences des écoles ne sont plus strictes. Pour départager les élèves, nous utilisons une *tie-breaking rule*, que nous décrirons ci-après. Nous avons préféré une approche *soft bounds* car elle est moins contraignante, plus simple à coder et elle respecte mieux les préférences des étudiants.

Student Deffered Acceptance-Single Tie Breaking-Soft Bounds (SDA-STB-SB)

1. On choisit une *tie-breaking rule* ;
2. Les élèves (ISEF et normaux) s'inscrivent provisoirement dans une école (la première de leur liste) ;
3. Tout d'abord, les écoles doivent remplir leur quota ISEF. Les élèves ISEF qui s'inscrivent dans une école sont acceptés en réserve, tant que le quota ISEF n'est pas atteint. Si celui-

ci est atteint, ne sont choisis que les élèves dont l'indice composite est le plus élevé et les élèves en trop rejoignent les élèves normaux qui ont fait une proposition à cette école, s'il y a des ex-aequo, les élèves sont départagés selon la *tie-breaking rule* établie au départ ;

4. Ensuite, les écoles remplissent la partie réservée aux élèves normaux, en les choisissant en fonction de leur indice composite, jusqu'à remplir toutes les places disponibles, les problèmes d'ex-aequo sont résolus grâce à la *tie-breaking rule*. Les autres élèves sont rejetés ;
5. A l'étape k, les élèves rejetés à l'étape $k-1$, s'inscrivent dans l'école suivante sur leur liste de préférences et l'on revient à l'étape 3 ;
6. L'algorithme se poursuit jusqu'à ce que tous les élèves soient dans une école, ou qu'ils aient été rejetés par toutes les écoles de leur liste. Dans ce dernier cas, les élèves sans école sont placés dans les écoles incomplètes ; car dans la réalité, nous ne pouvons pas laisser d'élève sans école (mais pour le cas théorique, ces élèves sont considérés sans école).

Dans cet algorithme, nous avons réuni à la fois des préférences non strictes et des quotas.

On sait que si les préférences sont strictes, le SDAA-SB donne un *matching* stable (Définition 2.4.7) et S-optimal et que ce mécanisme est *strategy-proof* (cf. Section 2.4). Cependant, nous allons montrer que l'introduction des préférences non strictes, n'est pas sans effet.

Avant d'aller plus loin, nous introduisons les notions suivantes :

Définition 5.3.1. *Pareto-dominance, Pareto-amélioration*

Un matching μ Pareto-domine un matching ν, si pour le matching μ, certains élèves ont une école qu'ils préfèrent à celle qu'ils ont dans ν et aucun élève n'a une école qu'il aime moins :

$$\forall s \in S,\ \mu(s) R_s \nu(s)\ et\ \exists s' \in S : \mu(s') P_{s'} \nu(s')$$

Un matching μ est une Pareto-amélioration d'un matching ν s'il Pareto-domine ce matching ν.

Pour le cas sans contrainte (voir [APR09]), Abdulkadiroğlu et al. montrent que si les préférences des écoles sont non strictes, la *tie-breaking rule* cause une perte d'efficience et la stabilité est artificielle (forcée). Il n'existe plus un unique *matching* S-optimal mais bien un ensemble de *matching* S-optimaux (qui ne sont Pareto-dominés par aucun autre *matching* stable, au sens de la Définition 2.2.3). On a donc que le SDAA muni d'une *tie-breaking rule* ne trouve pas toujours un *matching* qui est optimal pour les étudiants, bien qu'il soit stable et *strategy-proof*[4]. Nous ne discutons pas ici les différentes *tie-breaking rule* existantes, mais Abdulkadiroğlu et al. testent les règles de *tie-breaking*, pour le cas sans contrainte. Ils concluent que les *matching* produits par le SDAA muni d'une *tie-breaking rule* multiple (différente pour chaque école), qui ne peuvent pas être produits par le SDAA muni d'une *tie-breaking rule* unique (une seule règle pour toutes les écoles), alors ces *matching* ne sont pas S-optimaux (plus de détails dans [APR09]). Pour cette raison, nous utilisons une *single-tie-breaking rule* et

4. Le SDAA est *strategy-proof* pour n'importe quel profil de préférences des écoles et donc aussi pour n'importe quelle *tie-breaking rule*, cfr. [APR09].

nous appelons notre algorithme *Student Deffered Acceptance-Single Tie Breaking-Soft Bounds* (SDA-STB-SB). Nous avons choisi d'utiliser comme *tie-breaking rule*, un unique tirage aléatoire. Dans ce cas, on peut aisément étendre le résultat d'Abdulkadiroğlu et al. sur la perte d'efficience, au cas avec contraintes.

Remarque :

Muni du tirage aléatoire unique, le SDAA-SB ne donne pas toujours un *matching* S-optimal.

Cette remarque peut être illustrée par l'exemple suivant.

Exemple 5.3.1.

Prenons 4 écoles (A, B, C et D) et 8 élèves (désignés par un chiffre de 1 à 8), supposons que les préférences des élèves et des écoles sont les suivantes :

P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
A	A	B	A	B	C	C	C
D	B	A	D	C	B	A	D

TABLE 5.1 – Exemple 5.3.1, Préférences des élèves

$\succeq_A$	$\succeq_B$	$\succeq_C$	$\succeq_D$
3 7	6	4	1 2 3 4 5 6 7 8
1 2	2 5	5 7	–
4 5 6 8	3 1 4 7 8	8	–
–	–	6 1 2 3	–

TABLE 5.2 – Exemple 5.3.1, Préférences des écoles

Supposons qu'il n'existe que 2 types d'élèves : ISEF et normaux et que les élèves 1, 2, 5, 7 sont normaux et les autres ISEF.

Chaque école a une capacité maximale de 2 et les écoles A, B et C doivent au moins avoir un élève ISEF.

Tout d'abord, nous tirons aléatoirement un ordre pour classer les élèves : $3 \succ 4 \succ 1 \succ 2 \succ 5 \succ 7 \succ 6 \succ 8$. Ensuite, nous appliquons le SDAA-SB, et obtenons le *matching* final μ suivant :

$$\mu = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ D & B & A & D & C & B & A & C \end{pmatrix}$$

Mais ce *matching* est Pareto-dominé par le *matching* stable suivant :

$$\mu^* = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ D & A & A & D & B & B & C & C \end{pmatrix}$$

Nous avons montré que le SDAA-SB muni du tirage aléatoire unique, ne donne pas nécessairement un *matching* S-optimal.

Si nous revenons au cas sans contrainte, dans leur papier à propos de l'efficience [EE08], A. Erdil et H. Ergin proposent d'améliorer le *matching* obtenu en utilisant le *Stable Improvement Cycles Algorithm* (SICA), décrit dans la Section 5.4. Ils obtiennent ainsi un *matching* stable qui Pareto-améliore celui du SDAA-STB et est S-optimal. Cependant, dans [APR09], Abdulkadiroğlu et al. prouvent qu'il n'existe aucun mécanisme qui soit plus efficient que le SDA-STB et qui soit *strategy-proof*. Donc, l'amélioration apportée par le SICA est associée à la perte de la *strategy-proofness*.

On peut aisément généraliser le théorème sur la perte de la *strategy-proofness* d'Abdulkadiroğlu et al., au cas avec contraintes. Pour une *tie-breaking rule* t, on appelle DAA-SB^t, le SDAA-SB, qui utilise cette *tie-breaking rule* pour créer des préférences strictes.

Théorème 5.3.1. *Pour n'importe quelle tie-breaking rule t, il n'y a aucun mécanisme qui soit strategy-proof et Pareto-domine le matching obtenu par le SDAA-SB^t.*

La preuve est similaire à celle d'Abdulkadiroğlu et al. (voir [APR09]). Les seuls changements importants pour nous, sont repris dans le lemme suivant.

Lemme 5.3.1. *Supposons que ν domine μ (qui est le matching donné par le SDAA-SB^t), pour n'importe quel profil de performance. Alors, les mêmes élèves restent sans école, pour les deux matching.*

Démonstration.

S'il y a un élève s sans école dans le *matching* ν et pas dans μ, alors on a que $\nu(s) = sP_s\mu(s)$ car ν Pareto-domine μ. Alors $\mu(s)$ n'est pas acceptable pour s mais par construction du SDAA-SB, c'est impossible. Donc on a que $|\nu(c)| \geq |\mu(c)|, \forall c \in C$. Supposons qu'il existe une école $c \in C$ telle que $|\nu(c)| > |\mu(c)|$, cela signifie qu'il y a une place libre dans c pour le *matching* μ et qu'il existe un élève $s \in S$ tel que $\nu(s) = c \neq \mu(s)$. De plus, $cP_s\mu(s)$ puisque ν Pareto-domine μ. Comme on considère seulement l'approche *soft bounds*, s'il y a une place libre dans c et même si le quota maximal du type de s est dépassé, s devrait tout même être accepté dans c. Donc le *matching* μ ne respecte pas le critère de stabilité (*soft bounds*). Or μ est le résultat du SDAA-SB et donc par définition stable. Nous avons donc que $|\nu(c)| = |\mu(c)|$ et tous les étudiants sans école pour μ n'en n'ont pas non plus pour ν.

□

Notre algorithme mélange quotas et préférences non strictes. Nous avons montré qu'il y aura une perte d'efficience due aux préférences non strictes et qu'un mécanisme qui Pareto-améliore le *matching* obtenu par le SDAA-STB-SB n'est pas *strategy-proof*.

Pour nous remettre les idées en place, voici un petit tableau récapitulatif des différentes propriétés des algorithmes.

Propriétés \ Algorithme	SDAA	SDAA-STB + SICA	SDAA-SB	SDAA-STB-SB + GA
Préférences [5]	Strictes	Non strictes	Strictes	Non strictes
Quotas	Non	Non	Oui	Oui
Stable [6]	Oui	Oui	Oui	Oui
S-optimal	Oui	Oui	Oui	? ?
Strategy-proofness	Oui	Non	Oui	Non

TABLE 5.3 – Propriétés des différents algorithmes dérivés du SDAA.

A présent, nous pouvons passer à l'implémentation des algorithmes génétiques à proprement parler. Nous avons choisi 2 approches : la première consiste à créer des *matching*, sans imposer la stabilité et de les comparer à celui obtenu par le SDAA-STB-SB et la seconde approche vise à Pareto-améliorer le *matching* obtenu par le SDAA-STB-SB.

5.4 Algorithme génétique

Population

Dans ce problème, nous avons encore une fois besoin de définir les paramètres de l'algorithme génétique (voir Section 3.2).

- La population est un ensemble de chromosomes ;
- Un chromosome représente un *matching* ;
- Un gène représente un individu du groupe des étudiants ;
- Le locus, ou la place que le gène occupe dans le chromosome, représente l'école dans laquelle l'étudiant est placé. Cette école est déterminée directement par le nombre de places disponibles dans les différentes écoles (quotas).

Chaque étudiant est défini par un chiffre différent ainsi que par sa localisation géographique et son indice ISEF. De plus, il possède une liste de ses écoles préférées, dont la longueur varie.

Chaque école est aussi définie par un chiffre et par un nombre de places disponibles.

Fitness

Pour comparer le GA et le SDAA-STB-SB, nous avons choisi trois critères, qui peuvent prendre différentes pondérations et forment notre *fitness*. Ces trois critères sont :

- Les préférences des élèves ;
- La distance du domicile de l'élève à l'école ;
- Le respect des quotas ISEF.

5. Par préférences, on entend les préférences des écoles
6. La définition de stabilité diffère, selon qu'on considère ou pas les quotas

La première partie de la *fitness* est calculée en fonction du rang de l'école obtenue par l'étudiant.

$$u_s(\mu(s)) = \begin{cases} 1 - \dfrac{rk_s(\mu(s)) - 1}{|P_s|} & \text{si } \mu(s) \in P_s \\ 0 & \text{sinon.} \end{cases} \qquad \forall s \in S$$

Avec, $rk_s(\mu(s))$ est la place de l'école $\mu(s)$ dans la liste de préférences de s et $|P_s|$ le nombre de préférences de s.

Nous aurons donc que la première partie de la *fitness* du *matching* μ, est donnée par :

$$\mathcal{F}_1(\mu) = \frac{\sum_{s \in S} u_s(\mu(s))}{|S|}$$

La deuxième partie de la *fitness* est la somme normalisée des indices composites divisés par 2.

$$\mathcal{F}_2(\mu) = \frac{\sum_{s \in S} \frac{I_{s,\mu(s)}}{2}}{|S|}$$

Si le quota ISEF est respecté pour une école, elle obtient le score, $a_c = 1$ et sinon $a_c = 0$. La troisième *fitness* somme ces scores et les divise par le nombre d'écoles.

$$\mathcal{F}_3(\mu) = \frac{\sum_{c \in C} a_c}{|C|}$$

Comme nous disposons de plusieurs fonctions objectives, il existe deux approches possibles, décrites dans la Section 3.9.

En ce qui concerne l'approche a priori, nous obtenons la *fitness* suivante :

$$\mathcal{F}(\mu) = \alpha\mathcal{F}_1(\mu) + \beta\mathcal{F}_2(\mu) + (1 - \alpha - \beta)\mathcal{F}_3(\mu)$$

Avec $\alpha, \beta \geq 0$ et $1 - \alpha - \beta \geq 0$.

Pour le cas multi-objectif, chaque *fitness* est optimisée séparément (voir la Section 5.5) .

Sélection

La sélection des chromosomes se fait de la même manière que dans Section 3.5 sauf pour le cas du multi-objectif, auquel cas elle s'effectue comme dans la Section 3.9.

Crossover

Le *crossover* utilisé par la suite est le *crossover* PMX développé plus avant (la Section 4.4) car il est un des meilleurs sur le problème du mariage.

Mutation

La mutation est adaptée à notre problème et s'inspire de l'article de Erdil et Ergin ([EE08]). En effet, dans ce papier, les auteurs utilisent l'algorithme SDAA-STB et construisent des cycles pour améliorer le *matching* obtenu par le SDAA-STB. Le principe est simple, chaque élève qui n'a pas son premier choix pointe vers les écoles qui ont un rang plus élevé dans sa liste. Il ne peut y avoir une flèche d'un élève vers une école que si celui-ci a un ordre de priorité au moins égal à celui des élèves déjà présents dans l'école (noté l'élève $i \in D_c$, pour l'école c). Les élèves i_l appartiennent à un cycle de longueur n si $\forall l \in \{1..n\}$:

- $\mu(i_{l+1})P_{i_l}\mu(i_l)$;
- $i_l \in D_{\mu(i_{l+1})}$, avec $i_{n+1} = i_1$.

Ainsi, lorsqu'un cycle est formé, on met les élèves de ce cycle dans l'école de leur choix. Cette opération est itérée jusqu'à ce qu'il n'y ait plus moyen d'obtenir des cycles.

Nous avons adapté l'idée des cycles à la mutation mais ceux-ci sont aléatoires. Chaque mutation est un cycle :

1. On prend un élève i au hasard dans le *matching*, qui occupe la place k correspondant à une l'école b. Si b est fait partie des préférences de i et n'est pas son premier choix, on passe à l'étape 2, sinon, la mutation est terminée ;
2. On tire au hasard une école c ayant un rang plus élevé ;
3. Dans cette nouvelle école c, on présélectionne une place au hasard occupée par l'élève j. Si l'élève j préfère l'école b à l'école c ou bien si l'on est retombé sur l'élève occupant la place k, alors on passe directement à l'étape 6 sinon, on passe à l'étape 4 ;
4. Si l'école c fait partie des préférences de j, qu'elle n'est pas son premier choix et si l'élève i peut remplacer j dans c, en respectant les critères de stabilité, alors on place l'élève i à sa place dans l'école c, l'élève j devient l'élève i et on recommence à l'étape 2, sinon on recommence les étapes 2 à 4 pour un nombre de fois fini v, afin de trouver un élève j compatible. Si après les v itérations le cycle n'est toujours pas terminé, on passe à l'étape 5 ;
5. On place l'élève i dans la place de départ k et le cycle se termine (on saute l'étape 6) ;
6. Le cycle se termine et l'on échange de place les élèves de i et j.

Cette boucle n'est pas infinie du fait de la taille finie du problème et de l'introduction d'un critère d'arrêt v. Cependant, il est impossible de donner un nombre d'itérations fixe car il dépend des élèves qui sont tirés au hasard et de la place de leur école dans leurs préférences.

Afin de comparer les performances de nos algorithmes, nous avons effectué des simulations numériques, décrites dans la section suivante.

5.5 Résultats

Nous voudrions pouvoir répondre aux questions suivantes :

- Quelle est l'amélioration apportée par les algorithmes génétiques, est-elle significative ?

- Combien d'élèves sont affectés, combien ont eu une école qu'ils préfèrent (meilleur rang dans leur préférences) ?
- Peut-on obtenir un *matching* stable qui Pareto-améliore le *matching* du SDAA-STB-SB ?

Nous allons tenter de répondre à ces questions grâce aux simulations suivantes.

Fitness pondérée

Voici ce que nous avons obtenu comme *fitness* pondérée, en fonction de α et β, pour l'algorithme génétique et le SDAA. Comme l'algorithme génétique construit donne des résultats non déterministes et que la population de départ est tirée aléatoirement, les résultats sont une moyenne sur 50 itérations. Nous avons travaillé d'abord sur un petit échantillon de 400 élèves qui peuvent choisir 5 écoles parmi les 15 proposées. Les paramètres de l'algorithme génétique sont les suivants :

- $p_{crossover}$=0.5 ;
- $p_{mutation}$=0.9 ;
- Nombre d'étapes de l'algorithme=100 ;
- Taille de la population de départ=500 chromosomes.

Les écoles sont petites, moyennes ou grandes, c'est-à-dire, elles acceptent respectivement maximum 10, 20 ou 50 élèves afin de rester cohérentes avec le nombre d'élèves.

Remarque : le taux de mutation est très élevé dû à la configuration particulière de la mutation. L'abréviation SDAA-STB-SB est lourde et par la suite nous utilisons simplement SDAA pour y référer.

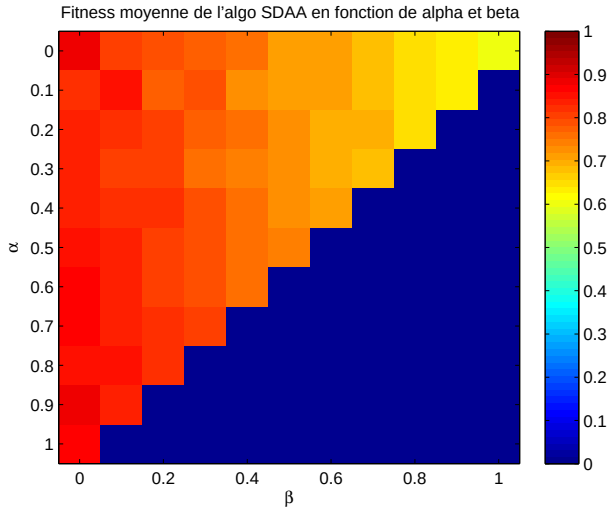

FIGURE 5.1 – Valeur moyenne de la *fitness* pondérée, en fonction de α et β, pour le SDAA (var=0.0022).

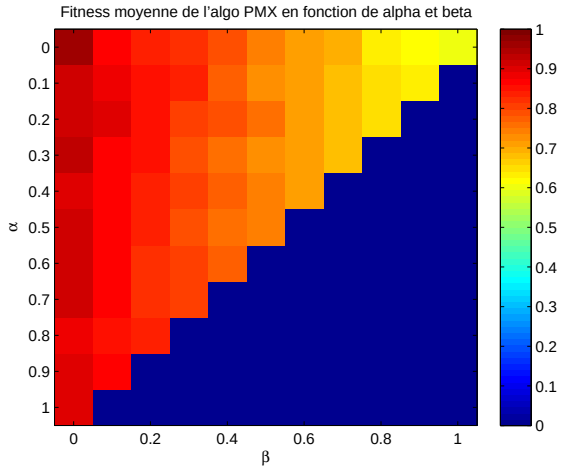

FIGURE 5.2 – Valeur moyenne de la *fitness* pondérée, en fonction de α et β, pour l'algorithme génétique PMX (var=0.0012).

Nous observons, que pour n'importe quelle valeur de α et de β, la *fitness* obtenue par l'algorithme génétique est meilleure que celle obtenue par le SDAA. On s'attend donc à ce qu'il y ait plus d'élèves qui aient leur premier choix, leur second choix, etc. Dès lors, voici le gain de rang de l'algorithme génétique sur le SDAA. Cela correspond à :

$$G = \sum_{i=1}^{6} (rg_{gen}(i) - rg_{SDAA}(i))$$

Avec, $rg_{gen}(i)$, le pourcentage d'élèves qui ont au moins leur $i^{ème}$ choix avec l'algorithme génétique et pareil pour le SDAA avec $rg_{SDAA}(i)$.

Voici la moyenne des gains obtenus correspondant à la Figure 5.2.

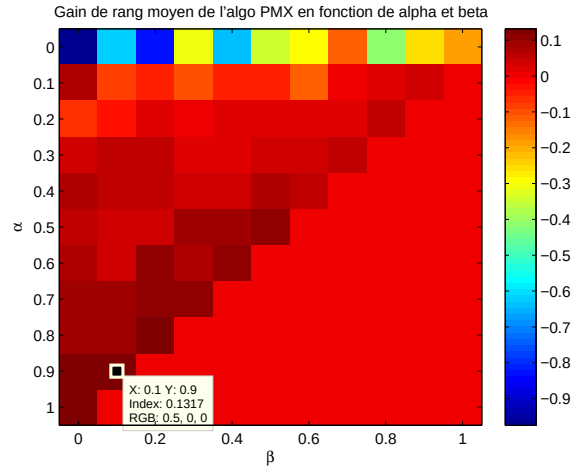

FIGURE 5.3 – Gain de rang du GA sur le SDAA, en fonction de α et β, avec une variance de 0.058.

Les résultats sont ceux espérés, nous obtenons un gain de rang positif à partir de $\alpha > 0.2, \quad \forall\beta$. Cela découle directement de la construction de la *fitness* pondérée. En effet, le coefficient α est associé au choix des élèves et donc plus α sera élevé et plus l'algorithme va sélectionner des chromosomes qui placent les élèves le mieux possible en fonction de leurs préférences. Par contre, lorsque α et β sont faibles, l'algorithme met l'accent sur les quotas de 20% d'élèves ISEF dans les écoles et donc ne tient pas vraiment compte des préférences des élèves. Le meilleur gain obtenu est lorsque $\alpha = 0.9$ et $\beta = 0.1$.

Pour mieux voir ce que cela donne pour un cas intermédiaire, voici le détail d'une simulation, $\alpha = 0.7$ et $\beta = 0.2$.

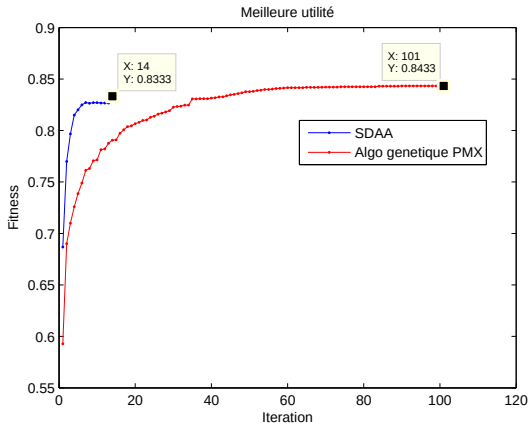

FIGURE 5.4 – Meilleure *fitness* pour $\alpha = 0.7$ et $\beta = 0.2$

Ce qui nous donne au niveau des rang obtenus par les élèves :

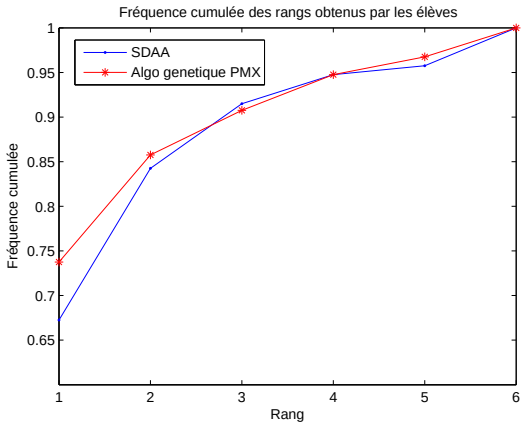

FIGURE 5.5 – Gain de rang du GA sur le SDAA, pour $\alpha = 0.7$ et $\beta = 0.2$

Avec un gain de 8.25% d'élèves qui ont un meilleur choix avec l'algorithme génétique plutot que le SDAA.

Le seul bémol à ce résultat est que les *matching* obtenus par l'algorithme génétique ne sont pas stables au sens de la Définition 2.4.7, avec une moyenne de 15.41% (variance = 0.018) des étudiants qui voudraient changer d'école. Cela reste une question ouverte, vaut-il mieux un *matching* stable, sans jalousie entre les élèves ou un *matching* qui a une utilité meilleure ? Cette question appartient au domaine économique et nous en reparlerons plus tard.

Le problème d'une *fitness* pondérée est qu'elle ne maximise pas les 3 objectifs en même temps mais seulement une pondération de ceux-ci. Dès lors, une composante peut être très mauvaise, voir deux lorsque $\alpha = 1$ ou $\beta = 1$ ou encore $\alpha = 0$ et $\beta = 0$. C'est pourquoi nous avons développé une autre méthode d'optimisation : le multi-objectif.

Mutli-objectif

D'abord, voici une simulation pour 400 élèves et 15 écoles (comme définis plus haut), avec les paramètres suivants :

- $p_{crossover}$=0.3 ;
- $p_{mutation}$=0.3 ;
- Nombre d'étapes de l'algorithme=150 ;
- Taille de la population de départ=700 chromosomes.

La fonction objectif 1 maximise les quotas d'élèves ISEF dans les différentes écoles, lorsqu'elle vaut 1, cela signifie que toutes les écoles ont bien 20% d'élèves ISEF. La fonction objectif 2 minimise pour chaque élève la distance de son domicile à l'école. Enfin, la fonction objectif 3 maximise les préférences des élèves. Ces 3 fonctions sont expliquées en détail dans la Section 5.4.

Le but de l'optimisation multi-objectifs est d'optimiser ces trois critères à la fois. En deux dimensions, nous ne pouvons représenter que deux fonctions objectifs et nous avons choisi de présenter la deuxième et troisième.

A la figure suivante, le point rouge représente l'utilité du SDAA et les autres points les meilleurs *fitness* à chaque itérations de l'algorithme génétique (rang 1).
Le SDAA obtient un *matching* qui a une utilité égale à (0.933, 0.603, 0.879), pour le SDAA, l'utilité du point noir vaut (0.933, 0.597, 0.890), du jaune (0.933, 0.621, 0.882) et du mauve (1.000, 0.621, 0.880). On constate que l'algorithme génétique obtient une meilleure *fitness* que le SDAA, dans les 3 dimensions. En noir, le meilleur point du point de vue de la fonction 3, en jaune le meilleur point pour la fonction 2 et en mauve le meilleur point pour la fonction 1.

De plus, le point mauve à lui seul dépasse l'utilité obtenue par le SDAA, dans toutes les directions, ce qui est une amélioration effective de l'utilité de toutes les parties : l'Etat qui voulait que les quotas soient respectés et que les élèves habitent le plus près possible de leur école ; et les élèves qui désiraient être dans l'école de leur choix.

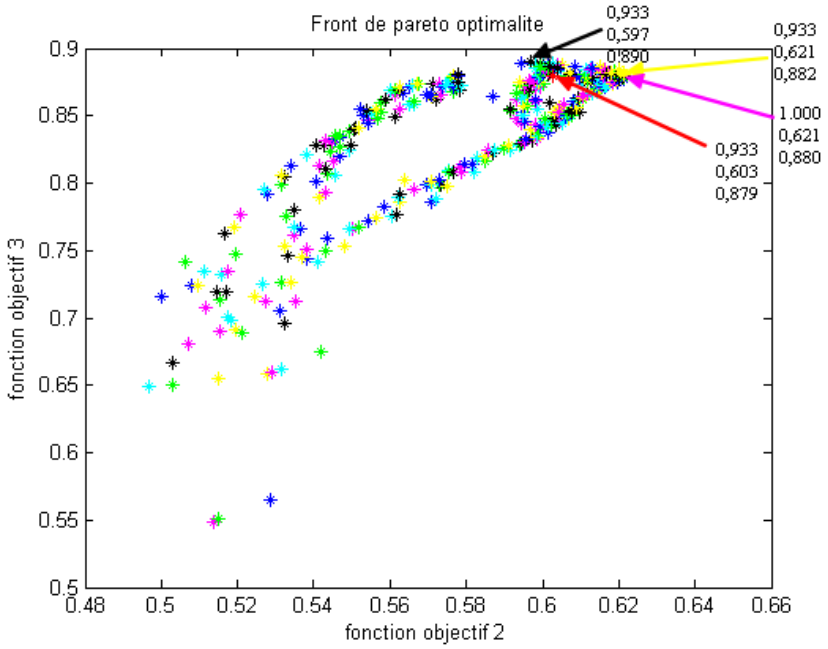

FIGURE 5.6 – Fronts de Pareto-optimalité, pour 400 élèves. Le point rouge reporte l'utilité du SDAA et les autres couleurs représentent les valeurs du front Pareto-optimal pour chaque génération de l'algorithme génétique.

Nous n'avons représenté graphiquement qu'une seule simulation car on ne peut pas moyenner un front Pareto-optimal. Cependant, 100 simulations ont été effectuées et voici les résultats : pour chaque composante de la *fitness*, combien de fois l'algorithme génétique trouve une valeur strictement plus élevée que celle du SDAA.

Composantes de la *fitness*	f_1	f_2	f_3
f_1	80	77	73
f_2	77	100	35
f_3	73	35	100

TABLE 5.4 – Cas multi-objectif : meilleure *fitness*. Ce tableau représente le nombre de fois que l'algorithme génétique trouve un meilleur *matching* que le SDAA, par composante de la *fitness* (sur 100 simulations).

On constate que notre algorithme génétique trouve presque toujours des *matching* qui ont au moins une des trois composantes de la *fitness* plus élevée que celles du *matching* obtenu par le SDAA (appelons le μ), respectivement dans 80% des cas pour la première composante et dans 100% pour la deuxième et troisième composante de la *fitness*. De plus, dans 77% des cas, notre algorithme trouve un *matching* dont la première et deuxième composante sont plus élevées que celles de μ, dans 73% des cas, un *matching* dont la première et troisième composante sont plus élevées que celles de μ et dans 35% des cas ce sont la deuxième et troisième composante qui sont plus élevées. Enfin, dans 18% des cas, notre algorithme trouve un *matching* dont les trois composantes de la *fitness* sont plus élevées que celles de μ. Grâce à l'optimisation mutli-objectif, nous pouvons choisir quelles composantes de la *fitness* nous vou-

lons maximiser en priorité. Cependant, encore une fois, les *matching* trouvés par l'algorithme génétique ne sont pas stables et pourtant cette propriété est assez attrayante puisqu'elle signifie que personne n'aura envie de rompre le contrat, ou dans notre cas l'inscription scolaire. Nous avons donc essayé une autre approche qui consiste à améliorer le *matching* obtenu par le SDAA.

Pareto-improvement

Les *matching* que l'on obtient ne sont pas stables et si on pénalise la non-stabilité, en posant la *fitness* des *matching* instables, égale à zéro, nous n'obtenons aucun résultat car il n'est pas facile de créer un *matching* stable aléatoirement et encore moins lorsque le nombre de *matching* possibles augmente (exponentiellement proportionnel au nombre d'élèves). C'est pourquoi, nous avons décidé de partir du *matching* obtenu par le SDAA (appelons le μ) et d'appliquer les GA pour voir si l'on peut obtenir une Pareto-amélioration. On force donc la stabilité, en posant la *fitness* des *matching* instables à 0 [7]. De plus, on ne tiendra compte que des *matching* qui Pareto-dominent μ : aucun élève ne doit avoir une école qu'il aime moins que celle qu'il a avec μ. Pour ce faire, nous pénalisons les *matching* qui ne Pareto-dominent pas celui du SDAA, en multipliant la *fitness* de ces derniers par une valeur inférieure à celle de la *fitness* de μ. Ainsi, nous sommes assurés que les seuls *matching* dont la *fitness* est plus élevée que celle de μ, sont stables et Pareto-dominent μ.

Nous avons utilisé l'algorithme génétique qui maximise seulement les préférences des élèves (*fitness* pondérée avec $\alpha = 1, \beta = 0$), avec les paramètres suivants :

- $p_{crossover}$=0.5 ;
- $p_{mutation}$=0.7 ;
- Nombre d'étapes de l'algorithme=15 ;
- Population initiale : 500 chromosomes (dont 100 fois le *matching* obtenu par le SDAA).

A la page suivante, se trouve une moyenne des résultats obtenus, sur 50 simulations, effectuées sur notre échantillon de 400 étudiants. Les résultats sont présentés dans un tableau qui s'inspire de celui d'Abdulkadiroğlu dans [APR09].

Même si l'amélioration apportée par les algorithmes génétiques est faible avec environ un peu plus d'un pourcent des élèves qui ont une meilleure école, elle s'accorde avec celle trouvée par Abdulkadiroğlu et al., avec l'algorithme SICA [8], pour le cas sans contrainte et avec préférences non-strictes [9]. En effet, en moyenne, 1.9 % des étudiants obtiennent une meilleure école grâce au SICA. Dès lors, si l'on considère que notre problème est plus compliqué, vu l'introduction des contraintes, nous pouvons dire que nos résultats sont cohérents. Quoi qu'il en soit, nous avons montré que le SDAA ne trouve pas un *matching* S-optimal et qu'il est possible d'améliorer le *matching* obtenu par ce dernier, en utilisant les algorithmes génétiques [10]. Malheureusement, ce gain d'efficience est accompagné de la perte de la *strategy-proofness*, par le Théorème 5.3.1. Nous ne discutons pas ici de ce problème car il n'est pas de notre ressort de juger de ce qui est préférable : la révélation des vraies préférences ou une plus grande

7. Si l'on part d'un *matching* stable, on peut pénaliser fortement les *matching* instables, afin que les *matching* stables évoluent.
8. Développé par Erdil et Ergin, dans [EE08]
9. Voir [APR09].
10. Cette amélioration ne vaut que pour les étudiants car ils reçoivent une école qu'ils préfèrent.

efficience. Le seul défaut des algorithmes génétiques est qu'ils sont non déterministes et que leur convergence n'est pas assurée. Dès lors nous ne pouvons pas savoir si le *matching* final, obtenu grâce aux algorithmes génétiques est optimal pour les étudiants ou bien s'il serait encore possible de le Pareto-améliorer. Nous pouvons supposer que les GA convergent en probabilité vers un *matching* qui est S-optimal (par construction) mais dû à leur variabilité, nous ne pouvons établir de résultats théoriques.

Algorithme / Choix	SDAA-STB-SB	SDAA-STB-SB + GA	Amélioration	♯ étudiants
1	275.7143 (9.1969)	278.2245 (8.9774)	+1	1.8571 (1.6202)
2	50.3061 (5.6943)	50.2245 (6.0252)	+2	1.4082 (1.3057)
3	28.8776 (4.1665)	28.3265 (4.2982)	+3	0.6531 (0.8792)
4	17.4490 (4.6996)	16.5918 (4.6901)	+4	0.2653 (0.5692)
5	11.2857 (3.1820)	10.2653 (3.1277)	+5	0 (0)
Sans école	16.3673 (3.1271)	16.3673 (3.1271)	-	0 (0)
Total	400	400		4.1837 (2.8988)

TABLE 5.5 – Amélioration du SDAA-STB-SB grâce aux algorithmes génétiques. La première colonne représente le rang de l'école assignée par les algorithmes, la seconde colonne est la moyenne du nombre d'étudiants obtenant leur premier choix, second, etc, avec le SDAA, la troisième colonne est la moyenne du nombre d'étudiants obtenant leur premier choix, second, etc, avec le SDAA amélioré par le GA, la quatrième colonne représente le nombre de places gagnées dans la liste des préférences des étudiants et enfin la cinquième colonne représente la moyenne des étudiants qui reçoivent une meilleure école. La déviation standard est reportée entre parenthèses [12].

A présent que nos méthodes fonctionnent bien, nous avons décidé de voir si elles donnent encore de bons résultats lorsque le nombre d'élèves augmente. A la page suivante, se trouve la moyenne des résultats obtenus sur 100 simulations, pour la population de Namur (1750 élèves), avec les paramètres suivants :

- $p_{crossover}$=0.5 ;
- $p_{mutation}$=0.4 ;
- Nombre d'étapes de l'algorithme=20 ;
- Taille de la population de départ=800 chromosomes (dont 400 fois le *matching* obtenu par le SDAA).

12. Cette table s'inspire de celle présentée dans [APR09] pour montrer l'amélioration du SICA (cas sans contrainte).

Algorithme / Choix	SDAA-STB-SB	SDAA-STB-SB + GA	Amélioration	♯ étudiants
1	1245.3 (27.4963)	1254.3 (27.4084)	+1	7.30 (1.8007)
2	222.55 (16.8369)	221.85 (16.5642)	+2	3.80 (1.9746)
3	130.20 (11.7748)	127.55 (11.4877)	+3	2.45 (1.4381)
4	69.40 (7.9671)	65.85 (8.5474)	+4	0.75 (0.7017)
5	37.25(5.2211)	35.10(4.9329)	+5	0 (0)
Sans école	45.35 (6.3347)	45.35 (6.3347)	-	0 (0)
Total	1750	1750		14.30 (2.6034)

TABLE 5.6 – Amélioration du SDAA-STB-SB grâce aux algorithmes génétiques, 1750 élèves. La première colonne représente le rang de l'école assignée par les algorithmes, la seconde colonne est la moyenne du nombre d'étudiants obtenant leur premier choix, second, etc, avec le SDAA, la troisième colonne est la moyenne du nombre d'étudiants obtenant leur premier choix, second, etc, avec le SDAA amélioré par le GA, la quatrième colonne représente le nombre de places gagnées dans la liste des préférences des étudiants et enfin la cinquième colonne représente la moyenne des étudiants qui reçoivent une meilleure école. La déviation standard est reportée entre parenthèses.

Encore une fois, on obtient une Pareto-amélioration du SDAA-STB-SB grâce aux algorithmes génétiques. Nous avons montré que nos méthodes fonctionnent correctement sur des échantillons plus grands. Le nombre d'élèves a quadruplé mais l'algorithme génétique fonctionne sans même doubler le nombre de chromosomes initiaux. Les algorithmes génétiques ont prouvé leur utilité dans le domaine du *matching* puisqu'aucun mécanisme n'avait encore été développé pour le cas avec contraintes et préférences non-strictes. De plus, nous ne nous sommes pas contentés de fournir un algorithme théorique, nous avons implémenté ces algorithmes et obtenus des résultats pratiques. Cette Pareto-amélioration est particulièrement appréciée d'un point de vue économique et les résultats ci-dessus ainsi que ceux du chapitre suivant ont été développés durant le stage de recherche à Padoue.

5.6 Limites et performances du modèle

Tout d'abord, les *matching* que nous obtenons grâce aux algorithmes génétiques peuvent améliorer celui obtenu par le SDAA (nous avons exploré différentes voies). Cependant, il est impossible de prouver s'il est S-optimal ou non, on peut juste le supposer d'après la convergence de l'algorithme. Dans le cas de la Pareto-amélioration, nous avons une perte de la *strategy-proofness* mais étant donné que les GA sont un procédé aléatoire, il est difficile de savoir comment on pourrait tricher sur ses préférences pour obtenir une meilleure école. On peut donc conclure que les algorithmes génétiques sont difficiles à manipuler. Encore une fois, nous ne pouvons pas juger quel *matching* est le meilleur, stable et *strategy-proof* (SDAA), stable et plus efficient (SDAA+GA) ou encore juste plus respectueux du décret mais instable (GA).

Dans la réalité, les étudiants ne choisissent pas aléatoirement leurs écoles et il se peut que leur choix rencontre les contraintes de la communauté française (domicile proche de l'école) dès lors il sera plus facile de respecter les préférences des élèves et les exigences de la communauté française. Si nous avions eu des données réelles, nous aurions pu tester les performances des GA. Néanmoins, la plus grosse partie du travail est faite, il suffirait d'adapter les paramètres et de faire quelques simulations.

Finalement, nous avons pu constater que, dans le cas sans contrainte et avec préférences strictes (mariage) comme dans le cas des préférences non-strictes et avec contraintes, les algorithmes génétiques ont prouvé leur utilité et leur efficacité. Les résultats sont très encourageants, même s'il reste du chemin à parcourir, surtout dans la caractérisation théorique des *matching* obtenus.

Pour en revenir au cas belge, il y a différents problèmes que nous avons identifiés. Tout d'abord, l'absence d'une centralisation des demandes d'inscription complique énormément la procédure et c'est à cause de cela que les parents ne comprennent pas bien ce qu'ils doivent faire, entrainant l'inscription multiple de certains élèves. Bien sûr, une centralisation signifierait une perte d'autonomie de la part des écoles et les parents penseraient perdre tout contrôle sur l'inscription de leur enfant mais celle-ci permettrait la garantie de l'application des critères du décret et l'utilisation d'un algorithme unique qui assignerait au mieux les élèves dans les écoles. Nous conseillons d'utiliser l'algorithme d'Elhers et al., dans une approche *soft bounds*, avec une *tie-breaking rule* unique (SDAA-STB-SB) et ensuite d'appliquer notre algorithme génétique pour améliorer le *matching* obtenu.

Chapitre 6

Jeux hédoniques

"Jamais je ne voudrais faire partie d'un club qui accepterait de m'avoir pour membre."
Groucho Marx

Ce chapitre a été spécialement développé durant le stage de Padoue et fais partie du papier écrit en collaboration avec Timoteo Carletti et Antonio Nicolò.

Nous avons choisi de présenter une application des algorithmes génétiques à un autre problème de microéconomie, plus particulièrement à la formation de coalitions et de clubs. La formation de coalitions est un problème courant en économie, en politique et dans bien d'autres domaines encore. Par exemple, la répartition des employés d'une entreprise en différents groupes de travail, l'introduction d'un bien public et sa consommation à différents niveaux par les habitants ; les applications sont multiples et surtout très fréquentes. Le bien public peut être par exemple la construction d'une piscine dans un quartier, que seuls les habitants ayant participé financièrement auraient le droit d'utiliser. Dès lors plus il y a de personnes qui participent et moins la piscine coutera cher mais plus elle sera bondée, tout est une question de préférences. L'utilité d'un individu dépend du nombre de personnes qui utilisent la piscine et du montant qu'il doit investir dans celle-ci. Nous allons analyser plus en profondeur le cas des coalitions hédoniques, c'est-à-dire lorsque l'utilité d'un joueur (sa satisfaction, son bien-être) dépend seulement de la coalition dans laquelle il se trouve et non pas des autres coalitions. Chaque joueur a donc des préférences sur l'ensemble des coalitions possibles (auxquelles il peut appartenir). Malheureusement, ce genre de problème ne se résout généralement pas en un temps polynomial [1]. Cependant, si le domaine des préférences est restreint, les résultats sont intéressants. Nous avons donc choisi d'étudier le cas des préférences anonymes, où les joueurs ont seulement une préférence sur la taille de la coalition, ou les biens que cette dernière produit. Les préférences ne portent donc plus sur chaque joueur de la coalition. De plus, nous analysons en profondeur les préférences *single-peaked* qui dépendent d'un pic, telles que plus on est proche du pic, mieux c'est pour le joueur (principe de la température). Il faut bien se rendre compte que ce genre de préférences est plutôt restrictif mais est dû au manque de résultats intéressants lorsque les préférences sont générales. Passons maintenant à la modélisation du problème.

6.1 Définitions et notations

Les notations étant similaires à celles de la théorie du *matching*, nous passons rapidement en revue cette section. Toutes les définitions suivantes sont données par Bogomolnaia et al.

1. Pour plus de détails, consulter Aziz et al., [ABH11] et [ABS13]

dans [BJ02].

Soit un ensemble fini de n joueurs, $N = \{1, \ldots, n\}$, une **coalition** est un ensemble $S \subseteq \mathcal{P}(N)$ et une **partition** de N est un ensemble $\Pi = \{S_k\}_{k=1}^{K}$ tel que $S_k \subseteq N, \forall k \in \{1, \ldots, K\}$; $S_k \cap S_j = \emptyset, \forall k \neq j$(les coalitions sont disjointes) et $\cup_{k=1}^{K} S_k = N$ (les coalitions partitionnent N).

Le **profil de préférences** du joueur i est représenté par la relation d'ordre $\succeq_i$ (complète transitive et réflexive) sur l'ensemble $\{S_k \subseteq N : i \in S_k\}$, tel que $S_1 \succeq_i S_2$, signifie que i aime au moins autant S_1 que S_2 et $S_1 \succ_i S_2$, signifie que i préfère strictement S_1 à S_2. Soit la partition Π, $S_\Pi(i)$ dénote l'ensemble $S_k \in \Pi$ tel que $i \in S_k$.

Un jeu $(N, \succeq)$ est formé d'un ensemble de joueurs, N et d'un profil de préférences, $\succeq$.

Une partition Π est **Pareto-optimale** si elle n'est dominée par aucune autre partition :

$$\nexists \Pi' \text{ telle que } \forall i \in N, \quad S_{\Pi'}(i) \succeq_i S_\Pi(i) \text{ et } \exists j : S_{\Pi'}(j) \succ_j S_\Pi(j)$$

Une partition Π est **faiblement Pareto-optimale** s'il n'existe aucune partition dans laquelle tous les joueurs sont mieux :

$$\nexists \Pi' \text{ telle que } \forall i \in N, \quad S_{\Pi'}(i) \succ_i S_\Pi(i)$$

La stabilité est définie de différentes manières, nous en présentons deux ici.

Une partition Π est **Nash stable** si $\forall i \in N : S_\Pi(i) \succeq_i S_k \cup \{i\}, \quad \forall S_k \in \Pi \cup \{\emptyset\}$

Une partition Π est **Individuellement stable** si $\nexists i \in N$ et $S_k \in \Pi \cup \{\emptyset\}$ tel que $S_k \cup \{i\} \succ_i S_\Pi(i)$ et $\forall j \in S_k, S_k \cup \{i\} \succeq_j S_k$

Une coalition est **ouverte** si au moins un joueur peut y être ajouté, sans que les joueurs déjà présents ne soient moins bien (au sens des préférences) sinon, la coalition est dite **fermée**.

Lorsque les préférences sont aussi générales, il se peut qu'il n'existe aucune partition qui soit individuellement stable ou Nash stable. Pour bien en être conscient, voici un petit exemple, extrait de [BJ02].

Exemple 6.1.1.

Soit $N = \{1, 2, 3\}$ et soit les préférences des joueurs :

$\{1,2\}$	$\succ_1$	$\{1,3\}$	$\succ_1$	$\{1\}$	$\succ_1$	$\{1,2,3\}$
$\{2,3\}$	$\succ_2$	$\{2,1\}$	$\succ_2$	$\{2\}$	$\succ_2$	$\{1,2,3\}$
$\{3,1\}$	$\succ_3$	$\{3,2\}$	$\succ_3$	$\{3\}$	$\succ_3$	$\{1,2,3\}$

TABLE 6.1 – Exemple 6.1.1, Préférences des joueurs

Nous avons un cycle dans les préférences : le premier joueur préfère être dans une coalition avec le joueur 2 plutôt qu'avec le joueur 3, le deuxième joueur préfère être avec le joueur 3

plutôt qu'avec le 1 et enfin le joueur 3 préfère être avec le joueur 1 plutôt qu'avec le 2. Chaque joueur préfère rester seul que d'être dans une coalition tous ensembles.
Il n'existe aucune partition qui soit individuellement stable ou Nash stable. En effet, si on part de la partition suivante : $\Pi = \{\{1\},\{2\},\{3\}\}$*, alors le joueur 1 veut rejoindre le joueur 2, qui veut être avec le joueur 3, qui préfère être avec le joueur 1 et ainsi de suite.*

L'exemple précédent montre la nécessité de restreindre les préférences des joueurs afin de trouver des résultats positifs. Nous ne considérerons donc, que les préférences anonymes : les joueurs ne se soucient plus que de la taille de leur coalition (les biens que la coalition produit sont considérés comme proportionnels au nombre de joueurs présents dans celle-ci).

Les préférences d'un joueur sont **anonymes** si $\forall S_1, S_2 \ni i$,

$$\sharp S_1 = \sharp S_2 \text{ entraine que } i \text{ est indifférent entre } S_1 \text{ et } S_2$$

Où $\sharp S$ désigne le nombre de personnes dans S.

Cependant, même si les préférences des joueurs sont anonymes, il peut ne pas exister de partition individuellement stable. Bogomolnaia et al. démontre cette proposition sur un exemple avec 63 joueurs mais étant donné que cet exemple est assez long, nous conseillons le lecteur intéressé de lire [BJ02], page 221. En conclusion, pour avoir des résultats positifs, les préférences des joueurs sont encore restreintes aux préférences *single-peaked* : plus on se rapproche d'une valeur seuil et plus le joueur est content (principe de la température).

Les préférences d'un joueur i sont ***single-peaked*** sur l'ensemble $\{1,\dots,K\}$ si il existe un nombre p_i, appelé pic de i, tel que $\forall S_1, S_2 \ni i, \sharp S_1 = s_1$ et $\sharp S_2 = s_2$ avec $s_1, s_2 \in \{1,\dots,K\}$,

$$[s_1 < s_2 \leqslant p_i \text{ ou } s_1 > s_2 \geqslant p_i] \Rightarrow S_2 \succ_i S_1$$

Autrement dit, un joueur préfère les coalitions dont la taille se rapproche de son pic. Pour comparer deux valeurs, l'une plus petite que le pic et l'autre plus grande, il faut introduire une règle de comparaison (cfr. Section 6.2).

Dans leur papier, Bogomolnaia et al. introduisent les notions suivantes.

Ordered characteristics

Chaque coalition S est décrite par une caractéristique, $c(S)$, qui se situe entre 0 et $\sharp S$ et qui peut être vue comme la production d'un bien public par la coalition S et sa consommation par chaque joueur présent dans S. Si les joueurs ont des préférences *single-peaked*, celles-ci portent seulement sur la production $c(S)$, tel que $S \succeq_i S' \iff [c(S') < c(S) \leqslant p_i \text{ ou } c(S') > c(S) \geqslant p_i]$.
Un jeu hédonique a ses caractéristiques ordonnées si les joueurs ont des préférences *single-peaked* sur $c(S)$ et $c(S)$ satisfait les propriétés suivantes :

1. Si $c(S) < \sharp S$, alors $c(S) = p_i$, avec $i \in S$;
2. Si $i \notin S, j \notin S$, et $p_i \geqslant p_j$ alors $c(S \cup i) \geqslant c(S \cup j)$. De plus, si $c(S \cup i) > p_i$ alors $c(S \cup i) = c(S \cup j)$.

Consistance

Un jeu hédonique qui a ses caractéristiques ordonnées est consistant si

$$[\exists i, \exists S \in \mathcal{P}(N) \text{ tels que } c(S \cup i) = p_i < c(S) < \min_{j \in S} p_j] \Longrightarrow c(T \cup i) \leqslant p_i, \forall T \in \mathcal{P}(N)$$

A présent que les concepts sont posés, nous pouvons passer aux résultats.

6.2 Résultats

Le résultat suivant est donné par Bogomolnaia et al. dans [BJ02].

Théorème 6.2.1. *Si un jeu hédonique a ses caractéristiques ordonnées, alors il existe une partition individuellement stable et de plus s'il est consistant, il existe une partition individuellement stable et faiblement Pareto-optimale.*

Bogomolnaia et al. proposent également un algorithme pour trouver cette partition individuellement stable, que nous décrivons brièvement ci-dessous [2].

Algorithme de Bogomolnaia et al.

Avant de commencer, les joueurs sont ordonnés par ordre croissant de leur pic. Donc on a que $i \geq j \Longrightarrow p_i \geq p_j$.

1. La première coalition S_1 est formée en ajoutant le premier joueur n. Ensuite, on ajoute les joueurs $k < n$, tant que $p_k \geq c(\{k, .., n\})$ et $\{k+1, .., n\}$ est ouvert à k ;
2. La coalition suivante $S_k, k \geq 2$ est formée de la même manière que la première, si ce n'est que l'on commence avec le joueur ayant le plus haut pic et sans coalition. Lorsque plus aucun joueur ne peut entrer dans S_k, on regarde si certains joueurs ne pourraient pas changer de coalition. Pour $j = 1 : k-1$, si S_j est ouverte, alors on parcourt chaque joueur de S_k. Si un joueur $i \in S_k$ gagne à changer de coalition et que la coalition S_j lui est ouverte, alors on le change de coalition ($c(S_j \cup i) \succ_i c(S_k)$ et $c(S_j \cup i) \succeq_f c(S_j), \forall f \in S_j$). Lorsque toutes les coalitions et tous les joueurs ont été passés en revue et s'il reste une coalition S_j ouverte à des joueurs de S_k, alors les joueurs indifférents entre S_j et S_k, sont changés de coalition. Chaque fois qu'un joueur est changé de coalition, on regarde si des joueurs sans coalition ne pourraient pas être ajoutés à S_k, en respectant la règle d'acceptation ;
3. L'algorithme se termine lorsque tous les joueurs sont dans une coalition.

Cet algorithme se finit en un temps polynomial et donne une partition qui est toujours individuellement stable. Cependant, si la condition de consistance n'est pas vérifiée, il se peut que la partition obtenue ne soit pas optimale. Voici un petit exemple, donné par Bogomolnaia et al. qui illustre cette affirmation.

2. Vous trouverez plus de détails ainsi que la preuve du théorème ci-dessus dans [BJ02].

Exemple 6.2.1.

Soit un ensemble de 8 joueurs. Supposons que les joueurs 1 à 5 ont leur pic égal à 8 et les joueurs 6 à 8 ont leur pic égal à 4, ils préfèrent tous une valeur plus élevée que leur pic plutôt qu'une valeur moins élevée. La caractéristique de S est définie comme suit :

$$c(S) = \begin{cases} \sharp S & \text{si } \sharp S \neq 6 \\ \min\{\sharp S, \min_{i\in S} p_i\} & \text{sinon.} \end{cases}$$

L'algorithme de Bogomolnaia et al. donne la partition $\Pi = \{\{1,2,3,4,5\}(= S_1), \{6,7,8\}(= S_2)\}$*, avec* $c(S_1) = 5$ *et* $c(S_2) = 3$*. Or la partition* $\Pi^* = \{\{1,2,3,4,5,6,7,8\}(= S_1^*)\}$ *Pareto-domine la partition* Π *car* $c(S_1^*) = 8$*, que tous les joueurs préfèrent à 5 et 3.*

C'est ici que les algorithmes génétiques deviennent intéressants, car ils peuvent être utilisés pour essayer d'apporter une amélioration. Nous avons donc développé un jeu hédonique avec des caractéristiques ordonnées mais non consistant.

Jeu non consistant

Pour une coalition S, on détermine $c(S)$ comme suit :

$$c(S) = \begin{cases} \sharp S & \text{si } \sharp S < a \text{ ou si } \sharp S > b \\ \min\{\sharp S, \min_{i\in S} p_i\} & \text{sinon.} \end{cases}$$

Avec $1 \leq a < b < n$. De plus, les joueurs ont des préférences *single-peaked*. Ainsi, il est facile de montrer que ce jeu a ses caractéristiques ordonnées et est non consistant.

Caractéristiques ordonnées :

1. Si $c(S) < \sharp S$, alors $c(S) = p_i$, avec $i \in S$. Car si $c(S) < \sharp S$, on est dans le cas $a \leq \sharp S \leq b$ et donc $c(S) = \min\{\sharp S, \min_{i\in S} p_i\} < \sharp S$ donc $c(S) = \min_{i\in S} p_i = p_k$, avec $k \in S$;
2. Si $i \notin S, j \notin S$, et $p_i \geqslant p_j$ alors $c(S \cup i) \geqslant c(S \cup j)$. De plus, si $c(S \cup i) > p_i$ alors $c(S \cup i) = c(S \cup j)$.
   (a) Si $\sharp S + 1 < a$ ou $\sharp S + 1 > b$ alors $c(S \cup i) = c(S \cup j) = \sharp S + 1$;
   (b) Sinon, $c(S \cup i) = \min\{\sharp S, \min_{k\in S\cup i} p_k\}$, alors
      i. Soit $\min_{k\in S\cup i} p_k \geq \sharp S + 1$ et $\min_{k\in S\cup j} p_k \geq \sharp S + 1$ donc
      $$c(S \cup i) = c(S \cup j) = \sharp S + 1$$
      ii. Soit $\min_{k\in S\cup i} p_k \geq \sharp S + 1$ et $\min_{k\in S\cup j} p_k < \sharp S + 1$ donc
      $$c(S \cup i) = \sharp S + 1 > c(S \cup j) = \min_{k\in S\cup j} p_k$$
      iii. Soit $\min_{k\in S\cup i} p_k < \sharp S + 1$ et $\min_{k\in S\cup j} p_k < \sharp S + 1$ donc
      $$c(S \cup i) = \min_{k\in S\cup i} p_k \geq c(S \cup j) = \min_{k\in S\cup j} p_k$$
      car
      A. Soit $p_i \geq \min_{k\in S} p_k$ et $p_j \geq \min_{k\in S} p_k$ donc $c(S\cup i) = c(S\cup j) = \min_{k\in S} p_k$;

B. Soit $p_i \geq \min_{k \in S} p_k$ et $p_j < \min_{k \in S} p_k$ donc $c(S \cup i) = \min_{k \in S} p_k > c(S \cup j) = p_j$;

C. Soit $p_i < \min_{k \in S} p_k$ et $p_j < \min_{k \in S} p_k$ donc $c(S \cup i) = p_i \geq c(S \cup j) = p_j$.

De plus, si $c(S \cup i) > p_i$ alors $c(S \cup i) = c(S \cup j)$. La seule possibilité est que $\sharp S + 1 < a$ ou $\sharp S + 1 > b$ et $c(S \cup i) > p_i$, alors $c(S \cup i) = c(S \cup j) = \sharp S + 1$.

Jeu non-consistant :

Pour avoir $\exists i, \exists S, c(S \cup i) = p_i < c(S) < \min_{j \in S} p_j$, il suffit de prendre S tel que $\sharp S < \min_{j \in S} p_j$ et $a \leq \sharp S < b$ et i tel que $p_i < \sharp S + 1$. Mais si on prend T tel que $\sharp T > b$ alors $c(T \cup i) = \sharp T + 1 > b > \sharp S > p_i$ et on n'a pas la condition de consistance.

On sait donc qu'avec ce genre de jeu, l'algorithme de Bogomolnaia et al. ne trouvera pas toujours une partition optimale et dans ce cas, il sera possible d'améliorer celle-ci. Une Pareto-amélioration n'est pas nécessairement possible car elle dépend aussi de comment sont distribuées les préférences des joueurs. Passons à présent à la section sur les algorithmes génétiques.

6.3 Algorithmes Génétiques

Nous avons encore une fois adapté les algorithmes génétiques à notre nouveau problème. Premièrement, les chromosomes sont définis comme des partitions de l'ensemble des joueurs. Nous modélisons une partition comme un vecteur, chaque valeur dans une cellule représente le numéro de la coalition et le locus représente le joueur.

Si l'on prend un exemple avec 10 joueurs, cela donne :

Partition =

1	2	1	3	3	1	1	2	1	3

Les joueurs 1, 3, 6, 7 et 9 sont dans une coalition (premier ensemble), les joueurs 2 et 8 dans une autre et enfin les joueurs 4, 5 et 10 dans une troisième coalition.

Chaque joueur a une liste de préférence qui est *single-peaked*. Les pics sont tirés au hasard et ensuite les autres valeurs sont réorganisées aléatoirement de manière à respecter les préférences single-peaked (plus on est proche du pic et mieux c'est) mais aussi de façon à départager deux valeurs (l'une plus grande que le pic et l'autre plus petite : dans le cas où $S_1, S_2 \ni i, c(S_1) < p_i < c(S_2)$ ou bien $c(S_2) < p_i < c(S_1)$). Ci-dessous, un exemple de préférences *single-peaked*, pour $n = 10$.

P_i	6	7	8	5	9	4	3	10	2	1

TABLE 6.2 – Exemple de liste de préférences *single-peaked*, pour $n = 10$. Le pic est 6 et ensuite viennent les tailles de coalitions, par ordre de préférence.

La population initiale est créée aléatoirement, et on rajoute 100 exemplaires de la partition obtenue par Bogomolnaia et al. puisque notre but est de voir s'il est possible de l'améliorer.

Pour le croisement, nous avons pris le *crossover* classique, décrit à la Section 4.4 car nous n'avons plus la contrainte d'avoir toutes des valeurs différentes dans le vecteur.

La mutation est adaptée à notre problème spécifique et elle s'effectue comme suit. Dans un premier temps, changer un seul joueur d'une coalition à une autre ne va pas améliorer la partition car c'est déjà ce que fait l'algorithme de Bogomolnaia et al. Nous prenons donc aléatoirement un chromosome et dans celui-ci, nous tirons au hasard une coalition. Ensuite, tous les joueurs de cette coalition sont déplacés vers une autre coalition (tirée aléatoirement). Dans un deuxième temps, nous recherchons quand même une amélioration individuelle. Nous tirons aléatoirement un chromosome dans la population initiale, et dans celui-ci un joueur, que nous changeons de coalition. Ces deux étapes sont effectuées $n_{mutation}$ fois, nous rappelons que le nombre de mutations dépend de la probabilité de mutation, du nombre de chromosomes ($Taillepop$) et du nombre de joueurs :

$$n_{mutation} = \text{Bi}(Taillepop \times n, p_{mutation})$$

Il nous reste à définir une *fitness* à optimiser. Pour évaluer nos chromosomes, nous avons créé la fonction d'utilité suivante, qui respecte les préférences des joueurs.

Premièrement, on définit une fonction d'utilité pour chaque joueur :

$$u_i(S) = \begin{cases} 1 - \frac{r_i(\sharp S)-1}{n} & \text{si } i \in S \\ 0 & \text{sinon.} \end{cases}$$

Avec $r_i(\sharp S)$, la position de $\sharp S$ dans la liste de préférences de i. Ainsi, un joueur qui est dans une coalition dont la taille est égale à son pic, a une utilité égale à 1.

Deuxièmement, une fonction d'utilité globale, qui est la *fitness* :

$$u(\Pi) = \sum_{i=1}^{n} u_i(S_\Pi(i))$$

On vérifie ensuite si la partition est individuellement stable, si ce n'est pas le cas, on pénalise le chromosome en lui attribuant une *fitness* plus faible (la *fitness*, $u(\Pi)$, est multipliée par une valeur inférieure à la *fitness* de la partition obtenue par Bogomolnaia et al., disons $u(B)$). De cette manière, nous conservons les partitions instables pour la sélection, en espérant qu'elles donneront de meilleures partitions (à travers la mutation et le *crossover*) et nous nous assurons que seuls les chromosomes avec une *fitness* plus élevée que $u(B)$, sont stables. Il faut savoir que la fonction d'utilité employée ici est arbitraire et pourrait être remplacée si besoin.

A présent, nous pouvons passer aux simulations numériques.

6.4 Simulations numériques

Nous avons testé nos algorithmes génétiques sur le jeu non-consistant décrit plus avant, avec les paramètres suivants :

- $p_{crossover}$=0.4 ;
- $p_{mutation}$=0.1 ;
- Nombre de générations=10 ;
- Taille de la population de départ=300 chromosomes (dont 100 qui sont la partition trouvée par l'algorithme de Bogomolnaia et al.).

Nous avons effectué 100 simulations, pour un nombre de joueurs égal à 20 ($= n$), en faisant varier les paramètres a et b. Pour respecter nos contraintes, a varie de 1 à $n-2$ et b, de $a+1$ à $n-1$. Le but des algorithmes génétiques, dans ce cas, est de trouver une partition qui Pareto-domine celle trouvée par l'algorithme de Bogomolnaia et al. (appelons-la B). Une partition, Π **Pareto-domine** B, si certains joueurs préfèrent strictement Π à B et les autres joueurs sont indifférents entre les deux coalitions. De plus, cette partition doit être individuellement stable. Nous pénalisons les partitions qui ne Pareto-dominent pas B, en multipliant leur *fitness* par une valeur inférieure à celle de B. Ainsi, seules les partitions stables qui Pareto-dominent B ont une *fitness* plus élevée que celle-ci. Les résultats de nos simulations sont présentés dans la Table 6.3, qui donne le nombre de Pareto-améliorations trouvées, sur 100 simulations.

a \ b	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	2	11	3	8	6	5	2	2	3	2	0	1	0	0	0	0	0	0
2	-	9	2	6	3	4	6	4	1	2	1	0	0	0	0	0	0	0
3	-	-	4	4	7	7	3	4	3	1	2	0	2	3	2	1	0	2
4	-	-	-	2	4	4	1	2	4	3	3	4	1	2	2	1	3	2
5	-	-	-	-	0	7	3	2	1	3	1	0	1	3	0	1	4	3
6	-	-	-	-	-	2	2	3	2	3	4	3	3	2	0	0	1	2
7	-	-	-	-	-	-	2	1	5	1	2	3	1	0	0	1	1	4
8	-	-	-	-	-	-	-	4	1	2	3	2	2	0	0	1	1	1
9	-	-	-	-	-	-	-	-	1	2	1	1	1	0	1	1	0	2
10	-	-	-	-	-	-	-	-	-	3	3	2	2	1	2	3	3	0
11	-	-	-	-	-	-	-	-	-	-	4	4	0	4	2	4	1	6
12	-	-	-	-	-	-	-	-	-	-	-	3	2	3	1	2	0	2
13	-	-	-	-	-	-	-	-	-	-	-	-	3	4	1	2	2	3
14	-	-	-	-	-	-	-	-	-	-	-	-	-	3	3	2	2	2
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	3	3	6
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	4	7
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	1
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3

TABLE 6.3 – $n = 20$, Amélioration de l'algorithme de Bogomolnaia et al. : nombre de fois qu'une partition qui Pareto-domine celle de Bogomolnaia et al. a été trouvée (grâce aux algorithmes génétiques), sur 100 simulations. La première ligne donne la valeur du paramètre b et la première colonne celle du paramètre a. Les autres lignes donnent, le nombre de Pareto-améliorations trouvées sur 100 simulations, en fonction des paramètres a et b.

a \ b	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	2.50	1.45	3.33	4.50	4.17	4.60	2.00	7.50	2.00	2.00	0	2.00	0	0	0	0	0	0
	(0.71)	(0.52)	(4.04)	(4.96)	(3.87)	(5.81)	(0)	(7.78)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
2	-	1.78	2.00	2.00	4.00	3.50	2.00	5.00	2.00	2.00	13.00	0	0	0	0	0	0	0
	-	(1.30)	(0)	(0.63)	(3.46)	(3.00)	(0)	(6.00)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
3	-	-	3.00	2.00	4.86	3.29	1.33	4.25	5.00	2.00	1.00	0	1.00	1.00	1.00	1.00	0	1.00
	-	-	(2.00)	(0)	(4.56)	(4.31)	(0.58)	(5.85)	(5.20)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
4	-	-	-	1.50	4.75	4.75	2	12.50	1.75	6.00	2.00	1.50	2.00	1.50	1.50	2.00	1.33	1.50
	-	-	-	(0.71)	(5.50)	(5.50)	(0)	(0.71)	(0.50)	(6.93)	(0)	(0.58)	(0)	(0.71)	(0.71)	(0)	(0.58)	(0.71)
5	-	-	-	-	0	3.29	10.00	1.50	2.00	9.67	2.00	0	2.00	2.67	0	2.00	2.50	2.33
	-	-	-	-	0	(3.40)	(6.93)	(0.71)	(0)	(6.81)	(0)	(0)	(0)	(0.57)	(0)	(0)	(0.58)	(0.58)
6	-	-	-	-	-	2.00	2.50	6.00	3.00	6.00	5.00	2.33	2.67	2.50	0	0	8.00	2.00
	-	-	-	-	-	(0)	(0.71)	(6.08)	(1.41)	(6.08)	(6.00)	(0.58)	(1.15)	(0.71)	(0)	(0)	(0)	(0)
7	-	-	-	-	-	-	3.50	2.00	4.20	3.00	2.00	4.33	3.00	0	0	2.00	2.00	3.00
	-	-	-	-	-	-	(2.12)	(0)	(3.03)	(0)	(0)	(3.21)	(0)	(0)	(0)	(0)	(0)	(2.00)
8	-	-	-	-	-	-	-	6.25	11.00	8.00	10.33	2.00	7.50	0	0	2.00	2.00	2.00
	-	-	-	-	-	-	-	(8.50)	(0)	(1.41)	(9.07)	(0)	(0.71)	(0)	(0)	(0)	(0)	(0)
9	-	-	-	-	-	-	-	-	8.00	2.00	6.00	2.00	12.00	0	8.00	7.00	0	2.50
	-	-	-	-	-	-	-	-	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0.71)
10	-	-	-	-	-	-	-	-	-	6.67	9.67	3.00	3.00	8.00	2.00	7.67	7.00	0
	-	-	-	-	-	-	-	-	-	(7.23)	(6.66)	(1.41)	(1.41)	(0)	(0)	(2.08)	(2.65)	(0)
11	-	-	-	-	-	-	-	-	-	-	2.25	3.50	0	3.50	5.00	3.25	2.00	4.00
	-	-	-	-	-	-	-	-	-	-	(0.50)	(3.00)	(0)	(2.38)	(2.83)	(2.50)	(0)	(2.76)
12	-	-	-	-	-	-	-	-	-	-	-	14	2.00	3.33	2.00	3.50	0	2.00
	-	-	-	-	-	-	-	-	-	-	-	(10.39)	(0)	(2.31)	(0)	(2.12)	(0)	(0)
13	-	-	-	-	-	-	-	-	-	-	-	-	8.67	5.75	12.00	2.50	11.00	5.33
	-	-	-	-	-	-	-	-	-	-	-	-	(8.33)	(4.79)	(0)	(0.71)	(11.31)	(3.51)
14	-	-	-	-	-	-	-	-	-	-	-	-	-	3.67	6.67	8.00	7.50	8.00
	-	-	-	-	-	-	-	-	-	-	-	-	-	(2.89)	(7.23)	(8.48)	(7.78)	(8.43)
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5.75	2.00	7.00	6.50
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	(7.50)	(0)	(8.67)	(6.25)
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2.83	3.25	5.71
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	(2.04)	(1.89)	(6.05)
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2.00	3.00
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	(0)	(0)
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2.00
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	(0)

TABLE 6.4 – $n = 20$, Amélioration de l'algorithme de Bogomolnaia et al., grâce aux algorithmes génétiques : Moyenne du nombre de joueurs qui ont une meilleure coalition, lorsqu'il y a une amélioration, en fonction des paramètres a et b. La déviation standard est reportée entre parenthèses.

On constate que l'algorithme génétique trouve souvent une Pareto-amélioration et que le nombre d'améliorations trouvées, sur 100 simulations, varie de 1 à 11. C'est un résultat assez positif puisque nous pouvons améliorer un algorithme existant, en trouvant une partition qui est strictement préférée par certains joueurs, sans que les autres soient lésés.

Nous pouvons caractériser ces améliorations, par le nombre de joueurs qui sont dans une coalition qu'ils préfèrent. La Table 6.4 donne la moyenne du nombre de joueurs qui obtiennent une coalition préférée à celle qu'ils avaient dans la partition de Bogomolnaia et al. Cette moyenne n'est calculée que sur les Pareto-améliorations trouvées.

Sur toutes les améliorations rencontrées, nous avons une moyenne de 4.02 joueurs qui se trouvent mieux dans leur nouvelle partition, ce qui fait 20.10% du nombre total de joueurs et ce n'est certainement pas négligeable.

Malheureusement, l'inconvénient, encore une fois, est l'impossibilité de caractériser théoriquement la partition obtenue par les algorithmes génétiques. On ne sait pas si la partition est Pareto-optimale ou s'il est encore possible de l'améliorer. De plus, selon un théorème de Aziz et al. (dans [ABH11]), vérifier si une partition est Pareto-optimale, dans le cas des préférences anonymes, est non-polynomial et il donc très couteux, lorsque la taille du problème est très grande, de vérifier si la partition obtenue par les GA est optimale ou non. Cependant, nous avons montré que, grâce aux algorithmes génétiques, on peut facilement trouver une amélioration. De plus, la partition finale est stable, ce qui signifie que personne n'a envie de changer de coalition. La stabilité est une propriété fort attrayante car elle garantit une certaine longévité de la partition obtenue. Si les contrats ou, dans notre cas, les coalitions se forment de manière décentralisée, il n'y a aucune garantie que la partition formée sera stable. Dès lors, l'utilisation d'un mécanisme qui amène une partition stable, justifie l'utilisation d'une *central house* qui centralise les préférences et donne une partition stable.

Nous avons également travaillé sur 50 et 100 joueurs, en changeant quelque peu les paramètres de l'algorithme génétique. Il n'est pas possible d'afficher, ici, tous les résultats obtenus, cependant, nous les donnons, pour $n = 50$, $a = 1$ et b variant de 2 à 31 avec une probabilité de *crossover* de 0.4, de mutation de 0.05 et 700 chromosomes dans la population initiale. Le choix de prendre les paramètres $a = 1$ et de faire varier b de 2 à 31 est arbitraire, étant donné que toutes les lignes ont la même allure que celle présentée ci-dessous. D'ailleurs, il est inutile de faire varier b jusque 50 car lorsque l'écart entre a et b est trop important, l'algorithme génétique ne trouve par de Pareto-amélioration (comme dans la Table 6.3).

a \ b	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	5	5	5	2	2	9	3	1	5	1	5	2	3	4
a \ b	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	3	4	5	5	6	5	5	8	2	3	1	2	0	0	0

TABLE 6.5 – $n = 50$, Amélioration de l'algorithme de Bogomolnaia et al. : nombre de fois qu'une partition qui Pareto-domine celle de Bogomolnaia et al. a été trouvée (grâce aux algorithmes génétiques), sur 100 simulations. La première et troisième ligne donnent la valeur du paramètre b et la première colonne celle du paramètre a. Les autres lignes donnent, le nombre de Pareto-améliorations trouvées sur 100 simulations, en fonction des paramètres a et b.

Cette table nous montre que même lorsque le nombre de joueurs augmente, les algorithmes génétiques trouvent des Pareto-améliorations à l'algorithme de Bogomolnaia. Or, plus le nombre de joueurs augmente et plus le nombre de partitions possibles augmente. Le nombre de partitions possibles, pour n joueurs est décrit par le nombre de Bell, i.e. $B_n = \frac{1}{e}\sum_{k=0}^{\infty}\frac{k^n}{k!}$, soit pour n=50, $B_{50} \approx 10^{47}$. Seulement pour $a = 1$, nous avons, sur les Pareto-améliorations trouvées, une moyenne de 8.56 joueurs qui obtiennent une coalition préférée à celle de Bogomolnaia et al., soit 17.13% de la totalité des joueurs.

Nous montrons dans le tableau suivant que les algorithmes génétiques fonctionnent toujours, lorsque le nombre de joueurs augmente (n est posé à 100, $a = 1$ et b varie de 2 à 61). Encore une fois, ce choix de paramètres est arbitraire étant donné que les résultats ont une structure similaire.

a \ b	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	4	4	1	3	1	1	3	4	4	0	1	1	4	1
a \ b	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	1	0	2	1	2	3	3	0	1	3	1	4	0	1	1
a \ b	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
1	3	2	0	2	2	3	3	2	3	3	1	4	2	4	0
a \ b	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
1	1	2	1	4	3	1	0	0	2	0	0	0	0	0	0

TABLE 6.6 – $n = 100$, Amélioration de l'algorithme de Bogomolnaia et al. : nombre de fois qu'une partition qui Pareto-domine celle de Bogomolnaia et al. a été trouvée (grâce aux algorithmes génétiques), sur 100 simulations. La première, troisième, cinquième et septième ligne donnent la valeur du paramètre b et la première colonne celle du paramètre a. Les autres lignes donnent, le nombre de Pareto-améliorations trouvées sur 100 simulations, en fonction des paramètres a et b.

Pour $n = 100$, le nombre de partitions possibles est environ de 10^{115} et pourtant, notre algorithme trouve celles qui Pareto-améliorent la partition de Bogomolnaia et al.. Sur nos simulations effectuées, nous avons en moyenne 16.52 joueurs qui voient leur utilité augmenter (meilleure coalition) par Pareto-amélioration trouvée. Ce chiffre n'est pas négligeable même s'il est légèrement inférieur à ceux trouvés lorsque le nombre de joueurs est moins élevé.

L'algorithme génétique décrit ci dessus peut également être utilisé pour maximiser l'utilité totale des participants, dans une approche utilitariste. Dans ce cas, nous trouvons quasi toujours une utilité plus élevée, par rapport à celle de la partition obtenue par l'algorithme de Bogomolnaia et al. Voici un exemple de maximisation de l'utilité, pour 100 joueurs ($a = 25$ et $b = 30$), avec les paramètres de l'algorithme suivants :

- $p_{crossover}$=0.4 ;
- $p_{mutation}$=0.2 ;
- Nombre d'étapes de l'algorithme=50 ;

– Taille de la population de départ=700 chromosomes (dont 100 qui sont la partition trouvée par l'algorithme de Bogomolnaia et al.).

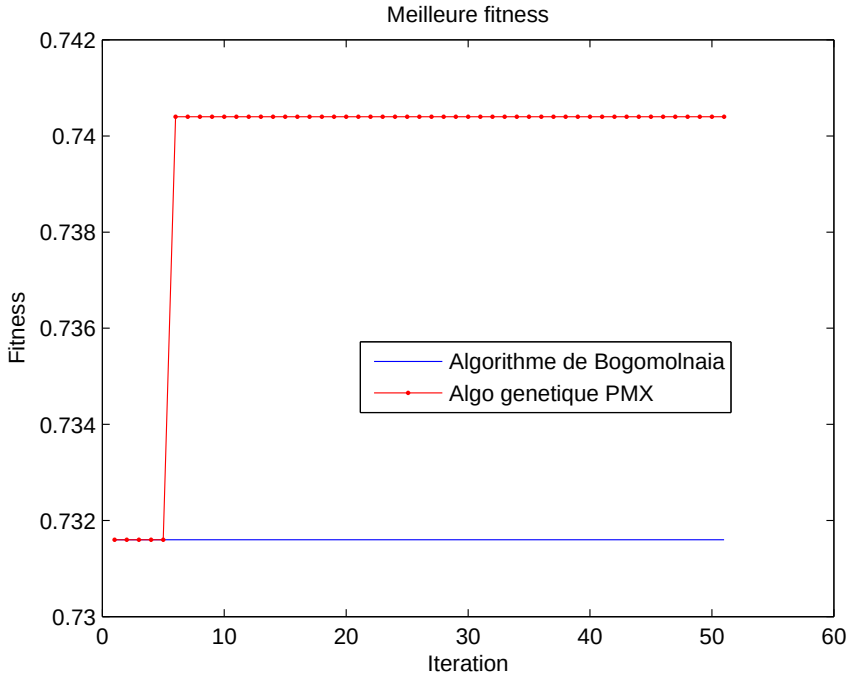

FIGURE 6.1 – Maximisation de la fonction utilitariste, $n = 100$ et avec les paramètres suivants : $a = 25$, $b = 30$. En bleu, la *fitness* de la partition obtenue par l'algorithme de Bogomolnaia et al. et en rouge, la *fitness* de la partition obtenue par l'algorithme génétique.

Quel que soit le but recherché, améliorer quand c'est possible la partition obtenue par Bogomolnaia et al. ou maximiser la fonction utilitariste, l'algorithme génétique peut atteindre ces deux objectifs, sans problème.

Dans ce chapitre, nous avons montré qu'il existe des problèmes de microéconomie difficiles à résoudre et que, grâce aux algorithmes génétiques, on peut leur trouver une solution. Même si cette solution ne peut pas être caractérisée théoriquement, c'est une amélioration apportée aux algorithmes existants et celle-ci est non négligeable.

Chapitre 7

Conclusion

"All models are wrong but some are useful."
George Box

Dans ce mémoire, nous avons appliqué les mathématiques à un problème concret basé sur le cas du décret belge des inscriptions dans les écoles et avons choisi comme outil, les algorithmes génétiques. Mais le but n'était pas seulement d'appliquer les algorithmes génétiques à ce problème, encore fallait-il apporter une interprétation qui avait du sens dans ce contexte. C'est pourquoi, durant le stage l'Université de Padoue, nous nous sommes penchés sur l'intégration des résultats dans un contexte économique.

Nous avons appliqué les algorithmes génétiques à différents problèmes de *matching* et apporté un regard nouveau sur cette théorie, qui jusqu'alors n'utilisait ces derniers que pour modéliser les comportements d'individus dans un jeu donné. La théorie du *matching* est vaste et de nombreux résultats théoriques restent sans répercussion pratique vu la difficulté de trouver un algorithme non-polynomial. Les algorithmes génétiques appliqués à deux problèmes de *matching* apportent une amélioration significative aux algorithmes déjà existants. Il est vrai que les GA ne convergent qu'en probabilité mais ils sont néanmoins faciles à adapter à un problème donné ; il suffit de bien modéliser et coder ce dernier. En plus d'être une approche nouvelle aux problèmes de *matching*, les algorithmes génétiques sont plus généraux que les techniques déjà existantes dans cette théorie. Pour les appliquer à un nouveau problème, il suffit de modifier quelques composantes, comme nous l'avons montré en adaptant l'algorithme construit pour le problème belge, au problème des jeux hédoniques.

Evidemment, nous aurions pu aller plus loin dans l'analyse des performances des algorithmes génétiques, en faisant varier les paramètres. Cependant, cette analyse dépassait le temps imparti pour ce mémoire, d'autant plus que les algorithmes génétiques ont été utilisés comme outils et n'étaient pas le principal sujet de ce mémoire. Le but était de comprendre le mécanisme qui se cachait derrière le problème belge, de l'analyser et d'y apporter une solution pratique. Dans la continuation de ce mémoire, d'autres tests pourraient être effectués, sur le nombre optimal de préférences que les élèves devraient donner, sur le fait de changer le quota d'élèves ISEF dans les écoles, changer la *tie-breaking rule*, compléter notre population synthétique, etc.

Nous avons montré que les problèmes ne manquent pas pour appliquer les algorithmes génétiques, surtout dans le domaine de la microéconomie, où ils n'ont été que peu exploités. Une application pratique qui se rapproche assez bien de ce que nous avons développé dans ce mémoire serait d'utiliser notre algorithme dans les agences du Forem. Ne serait-ce pas

plus facile pour les entreprises et les demandeurs d'emploi si on leur trouvait directement l'employeur/employé qui leur convient le mieux ?

Bibliographie

[Abd05] Abdulkadiroğlu, A., College admission with affirmative action, *International Journal of Game Theory*, Vol. 33, No. 4, 2005, pp. 535-549 ;

[ABH11] Aziz, H. ; Brandt, F. ; Harrenstein, Paul, Pareto Optimality in Coalition Formation, *Algorithmic Game Theory, 4th International Symposium, SAGT 2011, Amalfi, Italy, October 17-19, 2011. Proceedings*, Springer-Verlag 2011, pp. 93-104 ;

[ABS13] Aziz, H. ; Brandt, F. ; Seedig, H. G., Computing partitions in additively separable hedonic games, *Artificial intelligence*, 195, 2013, pp. 316-334 ;

[AC99] Aldershof, B. ; Carducci, O. M., Stable Marriage and Genetic Algorithms : A Fertile Union, *Journal of Heuristic*,5, 1999, pp. 26-46 ;

[APR05] Abdulkadiroğlu, A. ; Pathak, P. A. ; Roth, A. E., The New York City High School Match, *American Economic Review*, Vol. 95, No.2, 2005, pp. 364-367 ;

[APR09] Abdulkadiroğlu, A. ; Pathak, P. A. ; Roth, A. E., Strategy-proofness versus efficiency in matching with indifferences : redesigning the NYC High School match, *American Economic Revieuw*, Vol. 99, No. 5, 2009, pp. 1954-1978 ;

[AS03] Abdulkadiroğlu, A. ; Sönmez, T., School Choice : A Mechanism Design Approach, *American Economic Revieuw*, Vol. 93, No.3, 2003, pp. 729-747 ;

[Ald10] Aldashev, G., Risk Decision and Strategy, "Notes de cours de l'année académique 2010-2011" ;

[BCK11] Biró, P. ; Cantillon, E. ; Kübler, D., Matching in Practice, http://www.matching-in-practice.eu/, 2011, consulté le 18 février 2013 ;

[BJ02] Bogomolnaia, A. ; Jackson, M. O. ; The stability of Hedonic Coalition Structures, *Games and Economic Behaviour*, 38, 2002, pp. 201-203 ;

[CG09] Cantillon, E. ; Gothelf, N., Quel enfant dans quelle école ? Réflexion sur la régulation des inscriptions scolaires en Belgique, *Proceedings of the 18th Congress of Belgian French-speaking Economists*, Nov. 2009, CIFoP Editions ;

[Dal13] Dalton, J., The basic algorithm for GA, http://www.edc.ncl.ac.uk/highlight/rhjanuary-2007g01.php, 2013, consulté le 04 janvier 2013 ;

[Deb03] Deb, K., Multi-objective evolutionary algorithm : Introducing bias among Pareto-optimal Solution, *Advances in evolutionary computing*, partI, 2003, pp. 263-292 ;

[EE08] Erdil,A. ; Ergin, H.,What's the Matter with Tie-Breaking ? Improving Efficiency in School Choice, *American Economic Review*, American Economic Association, Vol. 98(3), Juin 2008, pp 669-689 ;

[Ehl10] Ehlers, L., School Choice with Control, Working Paper (CIREQ Cahier 13-2010), 2010, University of Montreal ;

[EHYY12] Elhers, L. ; Hafaldir, I. E. ; Yenmez, M. B. ; Yildirim, M. A., School Choice with Controlled Choice Constraints : Hard Bounds versus Soft Bounds, Working Paper Series, 2012, University of Montreal ;

[FWB] Delporte, L., ENSEIGNEMENT ET RECHERCHE SCIENTIFIQUE Décret Inscription, http://www.inscription.cfwb.be/index.php?id=295&no_cache=1, consulté le 02 novembre 2012 ;

[Gof13] Goffin, J., 97% des enfants inscrits dans l'école secondaire de leur choix, *Le Soir*, http://www.lesoir.be/230100/article/actualite/belgique/2013-04-22/97-des-enfants-inscrits-dans-l-ecole-secondaire-leur-choix, 22 Avril 2013, consulté le 12 mai 2013 ;

[GS62] Gale, D. ; Shapley, L. S., College Admissions and the Stability of Marriage, *The American Mathematical Monthly*, Vol. 69, No. 1, 1962, pp. 9-15 ;

[Gol89] Goldberg, David E., *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley Publishing Company, Inc, United States of America, 1989 ;

[HRU06] Haruvy, E. ; Roth, A. ; Ünver, M. U., The dynamics of law clerk matching : An experimental and computational investigation of proposals for reform the market, *Journal of Economics Dynamics & Control*, 30, 2006, pp. 457-486 ;

[Mit98] Mitchel, M., *An introduction to genetic algorithms*, No 1, Massachussetts Institute of Technology Press paperback edition, Cambridge, 1998 ;

[Mor11] Moreno Garrido, L. J. B., Métodos Matemáticos para la Economia, "Notes de cours de l'année académique 2011-2012" ;

[OSH87] Olivier, I. M. ; Smith, D. J. ; Holland, J. R. C., A study of permutation crossover operators on the travelling salesman problem, *Genetic algorithms and their applications : Proceedings of the Second International Conference on Genetic Algorithms*, 1987, pp. 224-230 ;

[Pat11] Pathak, Parag A., The Mechanism Design Approach to Student Assignment, *Annual Review of Economics*, Vol. 3 (c), Sept. 2011, pp. 513-536 ;

[Rot85] Roth, A. E., The College Admission Problem Is Not Equivalent to the Marriage Problem, *Journal of economic theory*, 36, 1985, pp. 277-288 ;

[RS90] Roth, A. E. ; Sotomayor, M. A. O., *Two-Sided Matching, A study in game-theoretic madeling and analysis*, No. 18, Cambridge University Press, Cambridge, 1990 ;

[Seh06] Seshadri, A., A fast elitist multi-objective genetic algorithm : NSGA-II, http://church.cs.virginia.edu/genprog/images/2/2f/Nsga_ii.pdf, 2006, consulté le 12 décembre 2012 ;

[TFL] Dendien, J., Le Trésor de la Langue Française Informatisé, http://atilf.atilf.fr/dendien/scripts/tlfiv5/advanced.exe?8;s=2590375680, consulté le 17 octobre 2012 ;

[Unv05] Ünver, M. U., On the survival of some unstable two-sided matching mechanisms, *International Journal of Game Theory*, 33, 2005, pp. 239-254 ;

[VN10] Ville de Namur, Namur en Chiffre, http://www.ville.namur.be/page.asp?id=1543, 2010, consulté le 28 décembre 2012 ;

[Decret11] Décret du Parlement de la Communauté française du 10.02.2011. portant modification du décret du 24 juillet 1997 définissant les missions prioritaires de l'enseignement fondamental et de l'enseignement secondaire (M.B. du 25.02.2011.).

Table des figures

Liste des tableaux

Annexe A

Mode d'emploi des codes

Dans cette annexe, nous décrivons brièvement les codes de ce mémoire et comment les utiliser. Tous les codes sont implémentés en Matlab et commentés en détail. A chaque problème correspond un dossier .rar. Nous passons seulement en revue les paramètres variables et comment utiliser les différents programmes.

Codes du Chapitre 4

Etant donné que nous avons effectué différents tests dans le Chapitre 4, nous avons plusieurs programmes principaux, chacun correspondant à un test. Pour le test de la stabilité, il faut utiliser le programme test_stabilite.m, pour le test sur les mutations, il faut utiliser le test_mutation.m et enfin pour le test sur le temps d'exécution, test_temps.m. Tous ces programmes ont les mêmes paramètres variables : $Nwomen$, le nombre de femmes, $Nmen$, le nombre d'hommes, $LprefW$, la longueur des préférences des femmes, $LprefM$, le longueur des préférences des hommes, *taillepop* la taille de la population de chromosomes, *nombreetapes*, le nombre d'étapes de l'algorithme génétique, *pmutation*, la probabilité de mutation, *pcrossover*, la probabilité de croisement. La variable *numero* correspond à un numéro donné à chaque méthode de *crossover* : 0 pour la méthode classique, 1 pour la méthode *ordercrossover*, 2 pour la méthode PMX, 3 pour la méthode des cycles et 4 pour le croisement circulaire. Pour le détail des sous-fonctions, référez-vous au commentaire en entête de celles-ci.

Codes du Chapitre 5

Fitness pondérée

Dans un premier temps, nous avons optimisé les différents critères du décret inscription. Pour ce faire nous utilisons le programme main.m, qui crée les préférences des étudiants, avec creationpreference.m et la population initiale de chromosomes, avec la fonction creationpop.m. Ensuite, il effectue le SDAA-STB-SB (SDAA.m) et l'algorithme génétique (algogen.m). Les paramètres variables sont $Nstudent$, le nombre d'étudiants, $Necole$, le nombre d'écoles, *taillepop*, la taille de la population de chromosomes, *nombreetapes*, le nombre d'étapes de l'algorithme génétique, *pmutation*, la probabilité de mutation, *pcrossover*, la probabilité de croisement et enfin, α et β, les paramètres de pondération des critères du décret. Attention, si le nombre d'élèves et d'écoles est variable, il faut tout de même d'abord créer les fichiers *fichier_population_ecoles_test.txt* et *fichier_population_eleves_test.txt*, grâce à

creation_population_eleves.m. Nous n'avons créé ces fichiers que pour 400 élèves et 15 écoles et pour 1750 élèves et 15 écoles. D'autres sous-fonctions ont été codées mais nous ne les détaillons pas ici, pour plus de précisions, consulter directement le commentaire de chaque fonction. Pour faire le test de la *fitness* en fonction des paramètres α et β, sur plusieurs simulations, nous utilisons le programme testalpha.m, avec les mêmes paramètres que ci-dessus. Ce programme écrit les résultats dans des fichiers et le programme graphes.m en fait la moyenne, présentée sous forme de graphes.

Multi-objectif

Pour l'optimisation mutli-objectif, nous avons utilisé en grande partie les codes décrits à la section précédente, à quelques modifications près. Nous utilisons le programme main.m qui d'une part, écrit dans un fichier, pour chaque simulation, les *fitness* finales des chromosomes de rang 1 (front de Pareto-optimalité) et d'autre part renvoie, pour une simulation, le graphe des *fitness* du SDAA et de l'algorithme génétique. Les paramètres variables sont $Nstudent$, le nombre d'étudiants, $Necole$, le nombre d'écoles, *taillepop*, la taille de la population de chromosomes, *nombreetapes*, le nombre d'étapes de l'algorithme génétique, *pmutation*, la probabilité de mutation et *pcrossover*, la probabilité de croisement. Pour obtenir le nombre de fois que l'algorithme génétique trouve une meilleure *fitness* que le SDAA, nous utilisons le programme graphes.m.

Pareto-improvement

Encore une fois, pour la Pareto-amélioration, nous avons utilisé les codes de la section *Fitness* pondérée, avec $\alpha = 1$ et $\beta = 0$ car nous ne nous intéressons qu'aux préférences des élèves. La seule fonction qui est modifiée est la fonction qui évalue la *fitness* (evaluation.m) car elle prend en compte le *matching* donné par le SDAA-STB-SB et vérifie s'il y a Pareto-amélioration ou pas. Le programme principal est le programme main.m et pour obtenir les moyennes, on utilise moyenne.m.

Codes du Chapitre 6

Le programme main.m crée les préférences des joueurs avec creationpreference.m et la population initiale de chromosomes, avec la fonction chromosome.m. Ensuite, il effectue l'algorithme de Bogomolnaia et al. (coalition.m) et l'algorithme génétique (algogen.m). Les paramètres variables sont n, le nombre de joueurs, *taillepop*, la taille de la population de chromosomes, *nombreetapes*, le nombre d'étapes de l'algorithme génétique, *pmutation*, la probabilité de mutation, *pcrossover*, la probabilité de croisement et enfin, a et b, les paramètres du jeu hédonique non-consistant. Ce programme utilise aussi une fonction pour calculer la *fitness* des partitions, evaluationstable.m (partitions stables et qui Pareto-dominent celle de Bogomolnaia). Nous avons également codé diverses petites fonctions intermédiaires, telle que celle calculant la caractéristique d'une coalition S ($c(S)$) : publicgood.m. L'algorithme génétique utilise principalement les fonctions suivantes : selection.m, crossover.m, crossoverpopulation.m et mutationoptimal.m, pour effectuer respectivement la sélection, le croisement et la mutation ; mais nous ne les détaillons pas ici. Pour obtenir les tableaux présentés dans le Chapitre 6, on utilise results.m, avec les paramètres n, le nombre de joueurs et a et b, les paramètres du jeu hédonique non-consistant ; en fonction des fichiers créés par le programme principal.