

Génération et analyse automatique de simulations multi-agents

P. Caillou^a
caillou@lri.fr

^aLaboratoire de Recherche en Informatique,
Université Paris Sud, France

Résumé

Les simulations à base d'agents sont de plus en plus utilisées pour réaliser des études en sciences sociales. Toutefois, peu de méthodologies existent pour guider ces études. De nombreux problèmes cruciaux restent ouverts, tels que la détermination du nombre de simulations à effectuer ou la validation des résultats obtenus. Dans cet article, nous décrivons un outil et sa méthode de conception permettant de générer automatiquement de nouvelles simulations jusqu'à ce que les résultats obtenus et analysés automatiquement soient statistiquement valides. Les interdépendances entre variables sont analysées à l'aide du test du Chi-deux, nécessitant un minimum d'hypothèses. Le modèle décrit est générique et l'outil-application présenté peut être appliqué à n'importe quelle simulation utilisant la plateforme RePast. Un exemple d'application à une simulation du marché du travail académique français est présenté.

Mots-clés : Simulation multi-agents, Vérification et validation des systèmes multi-agents, Environnements de développements multi-agents

Abstract

Multi-agent based simulation (MABS) is increasingly used for social science studies. However, few methodologies exist. A strong issue is the choice of the number of simulation runs and the validation of the results by statistical methods. In this article, we propose a method to automatically generate and run new simulations until the results are statistically valid using a chi-square test. The choice of the test configuration allows both a general overview of the variable links and a more specific independence analysis. We present a generic tool for any RePast-based simulation and apply it on an Academic Labor Market economic simulation.

Keywords: Multi-Agent Based Simulation, Simulation Validation, Simulation Tool

1 Introduction

L'utilisation de simulations à base d'agents (SBA) est de plus en plus fréquente dans les travaux en sciences sociales. Leurs principaux avantages sont une définition naturelle des comportements au niveau micro (les agents) combinée à une observation des variables au niveau macro, et une définition très simple de scénarii alternatifs. Elles autorisent la réalisation d'analyses aussi bien positives que normatives de problèmes qui seraient trop complexes pour une analyse purement analytique[2, 14]. Toutefois, peu de travaux méthodologiques existent sur l'utilisation des SBA. La validation, en particulier, constitue un problème majeur pour la plupart des simulations : par définition, le problème simulé est trop complexe pour être validé analytiquement (sinon la simulation ne serait utilisée que comme source d'inspiration pour le chercheur). D'autres méthodes de validation doivent donc être considérées. Le résultat d'une simulation étant un ensemble de données (par exemple des temps d'évacuation pour une simulation d'attentat dans un stade), il peut sembler naturel d'utiliser des outils statistiques pour valider les résultats obtenus¹ (comme si ces données étaient issues d'une mesure empirique). Même si leur utilisation se popularise, la validation la plus utilisée reste toutefois le recours exclusif à l'expert [7]. Ce choix n'est pas obligatoirement idéologique ou méthodologique, il est souvent la conséquence d'un manque d'outils, de temps ou de compétences adaptées pour réaliser les tests. Notre but est de montrer comment concevoir un outil d'analyse automatique nécessitant un minimum de connaissances et d'interventions de l'utilisateur.

Pour pouvoir utiliser la majorité des tests sta-

1. Nous considérons ici la validation des résultats de la simulation en supposant le modèle valide, et non la validation des hypothèses du modèle par rapport à la réalité. Par exemple, des tests statistiques peuvent valider le fait que le facteur le plus important pour le temps d'évacuation est le nombre de sorties du stade. Toutefois les hypothèses du modèle - le comportement des agents et la modélisation du stade - doivent être validées séparément

tistiques, une des principales conditions est de disposer d'un échantillon de données suffisamment large. Par rapport aux données empiriques, pour lesquelles obtenir de nouvelles données peut être difficile voire impossible, la SBA a ici un avantage : il est toujours possible de générer plus de données dans la configuration voulue (même si le temps de calcul peut être important). Nous allons utiliser cet avantage pour générer automatiquement de nouvelles données jusqu'à ce que les résultats soient statistiquement valides.

Un outil idéal fonctionnerait avec n'importe quelle plateforme de simulation et choisirait automatiquement le test statistique le plus adapté en fonction de l'objectif de l'utilisateur. Dans un premier temps, nous allons débiter avec un seul test sur une seule plateforme. Nous avons choisi la plateforme Repast3 [9] du fait de sa grande popularité. Pour que l'approche soit la plus générique possible, l'interaction avec la plateforme est toutefois minimale : l'outil lit et écrit des fichiers de configuration et de résultats et lance le simulateur (la généralisation à d'autres plateformes se limite donc à une adaptation au nouveau format de fichier).

Un des premiers résultats de la majorité des SBA concerne les liens entre paramètres et variables. Pour mesurer ce lien, nous allons utiliser le test du Chi-deux (χ^2) de Pearson. Il vérifie si l'hypothèse d'indépendance entre deux paramètres/variables peut être rejetée ou non (en comparant la population observée à une distribution théorique obtenue en supposant l'indépendance). Ce test a l'avantage d'avoir un nombre très faible de conditions d'application.

L'objectif de ce papier est de proposer un modèle d'outil générique permettant, à partir de n'importe quelle simulation RePast, de générer automatiquement de nouvelles simulations jusqu'à ce que les résultats obtenus, analysés à l'aide de tests du Chi-deux, soient statistiquement valides. Pour illustrer notre modèle et l'outil associé, nous l'avons appliqué à une simulation du marché des maîtres de conférences en France (présenté - sans validation statistique - dans [4, 5]). Les paramètres principaux du modèle concernent les caractéristiques de la procédure de recrutement et les fonctions d'utilité des candidats et des universités. Les variables observées concernent en particulier les taux de recrutement globaux et locaux.

Dans la suite de cet article, nous allons tout d'abord présenter en section 2 l'état de l'art

dans le domaine des outils et des méthodologies de SBA. Puis nous allons décrire notre modèle, le test utilisé et les différentes applications possibles (section 3). La simulation utilisée comme application est décrite en section 4 et des exemples de résultats en section 5. Nous concluons en section 6.

2 Etat de l'art

Un grand nombre de problèmes économiques, géographiques ou sociaux ont été étudiés à l'aide de simulations à base d'agents [2, 11]. Plusieurs plateformes de développement sont disponibles, certaines plus accessibles, mais aussi plus fermées, telles que NetLogo[16], d'autres plus ouvertes telles que ModulEco[10] ou RePast[9] (voir une synthèse dans [12]). Toutefois, aucune de ces plateformes n'intègre de module de génération automatique avec critère d'arrêt ou d'analyse statistique. L'outil *BehaviorSpace* de NetLogo, en particulier, ne permet de choisir qu'un espace de valeurs à explorer pour les paramètres d'une simulation (c'est l'équivalent d'un fichier de configuration de RePast). L'outil le plus proche de ce que nous proposons est le "robot scientist"[13] développé pour réaliser automatiquement une série d'expériences biologiques de façon autonome. Notre approche est l'équivalent dans le domaine des SBA, en incluant les tests statistiques, une sorte de "robot stagiaire" pour aider le scientifique dans son étude...

Le projet SimExplorer[1] constitue un outil proche mais complémentaire au nôtre : le but de SimExplorer est de fournir un outil générique de gestion de l'historique des simulations réalisées avec les paramètres associés et de pouvoir lancer de nouvelles simulations. Ce projet, en cours de développement, n'a pour l'instant pas de notion de génération répétée, de critère d'arrêt ou d'analyse de résultat, il s'agit plus d'une interface générique et d'un outil de gestion de résultats. Une de nos perspectives consiste à intégrer notre approche de génération automatique et d'analyse en plug-in de SimExplorer.

La calibration et la validation ont toujours été des problèmes clefs des SBA. Peu de méthodologies ont été proposées du fait de la grande variété des cas possibles. Une classification des méthodes utilisées empiriquement a été réalisée dans [17, 8]. Par ailleurs, une étude chronologique et statistique des méthodes statistiques et à base d'expert utilisées de 1998 à 2008 a été réalisée par [7], montrant que l'utilisation

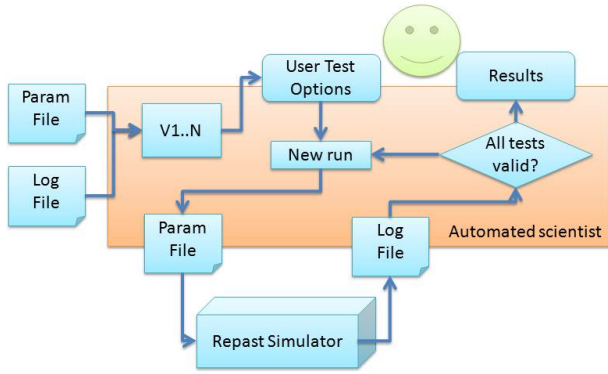


FIGURE 1 – Synthèse du modèle : à partir d'un fichier de configuration et de résultat, l'outil déduit les variables du modèle. L'utilisateur choisit la configuration, puis l'outil génère, lance et analyse de nouvelles simulations jusqu'à ce que les résultats soient valides, puis présente les résultats à l'utilisateur

d'outils statistiques reste marginale (inférieure à 10%), même si elle s'est accrue depuis 1998. Enfin, le test du Chi-deux est un des tests les plus couramment utilisés pour analyser les résultats des simulations du fait de son faible nombre d'hypothèses. Une introduction au test peut être trouvée dans [18] et une discussion plus approfondie dans [6].

3 Modèle

Dans cette partie, nous commençons par donner un aperçu de notre approche (3.1), avant de préciser les notations utilisées (3.2), puis de décrire le test du Chi-deux (3.3). Nous présentons ensuite en détail notre modèle (3.4), proposons des heuristiques pour accélérer la procédure (3.5) et discutons quelques objectifs atteignables à l'aide du modèle proposé (3.6).

3.1 Présentation

Le fonctionnement global de notre modèle peut être présenté simplement (voir une synthèse Fig. 1) : à partir d'une simulation RePast, l'outil déduit les variables et les paramètres utilisés et demande à l'utilisateur de choisir la configuration qui l'intéresse. De nouvelles simulations sont alors générées, lancées et analysées jusqu'à ce que tous les tests d'indépendance entre paramètres/variables soient valides. Enfin, les résultats des tests et leur marge d'erreur sont présentés à l'utilisateur.

3.2 Notations

Par la suite, nous utiliserons les notations suivantes :

- $V = \{v^1..v^{p+r}\}$ désigne l'ensemble des variables², comprenant :
 - $v^1..v^p$ pour les paramètres de la simulation.
 - $v^{p+1}..v^{p+r}$ pour les variables observées.
- $U \subseteq V$ désigne le sous-ensemble de variables choisi pour réaliser les tests d'indépendance (un test d'indépendance est réalisé pour chaque couple $v^i \times v^j$ avec $v^i \in U$ et $v^j \in U$).
- $x^1..x^{p+r}$ désignent les valeurs observées pour les variables $v^1..v^{p+r}$ lors d'une simulation.
- c^i désigne le nombre de classes utilisées pour la variable v^i .
- l_t^i et l_{t+1}^i désignent les bornes inférieures et supérieures de la classe t de la variable v^i .
- n . désigne le nombre total d'observations (et donc le nombre total de simulations réalisées).
- n_t^i désigne le nombre d'observations pour la classe t de la variable v^i .
- $n_{tt'}^{ij}$ désigne le nombre d'observations correspondant simultanément à la classe t de la variable v^i et à la classe t' de la variable v^j (effectif observé).
- $e_{tt'}^{ij}$ désigne l'effectif théorique correspondant simultanément à la classe t de la variable v^i et à la classe t' de la variable v^j (le nombre d'observations qui aurait été observé si les variables étaient indépendantes).
- Pr^{ij} désigne la probabilité pour que les variables v^i et v^j ne soient pas indépendantes selon le test du Chi-deux.
- $nbrun$ désigne le nombre de simulations par série de simulations (c'est-à-dire entre chaque mise à jour des résultats).

3.3 Test du Chi-deux

Le test d'indépendance du Chi-deux de Pearson s'utilise avec deux variables discrètes (mais rien n'empêche de discrétiser une variable continue en créant des classes de valeurs, automatiquement ou manuellement). Il teste la véracité de l'hypothèse H_0 : la variable v^1 est indépendante de la variable v^2 . En résumé, le test va vérifier si les lignes et les colonnes du tableau de contingence (distribution des observations) sont

2. Le test étant identique, nous étudions de la même façon l'indépendance des couples variable observée/variable observée et paramètre/variable observée. Nous utilisons donc la même notation et regroupons les deux notions sous le terme générique de variable.

TABLE 1 – Sexe des candidats et des admis à un poste de maître de conférences, France, 2007.

$n_{tt'}^{12}$	Hommes	Femmes	Total $n_{t'}^2$
Recrutés	1081	725	1806
Non recrutés	4234	3173	7407
Total n_t^1	5315	3898	9213

TABLE 2 – Distribution théorique

$e_{tt'}^{12}$	Hommes	Femmes	Total $e_{t'}^2$
Recrutés	1041,9	764,1	1806
Non recrutés	4273,1	3133,9	7407
Total n_t^1	5315	3898	9213

indépendantes. Pour cela, la distribution observée (le tableau de contingence) va être comparée à une distribution théorique qui aurait été obtenue si les variables étaient effectivement indépendantes. Si H_0 est vraie, alors $P(v^1 \cup v^2) = P(v^1) \times P(v^2)$.

Par exemple, nous allons tester si le sexe du candidat (v^1) a un impact sur la probabilité d'être recruté à un poste de maître de conférences (v^2). Les données sont présentées dans la Table 1. Les variables sont des variables discrètes avec deux classes chacune, $c^1 = c^2 = 2$. A partir de cette distribution observée, nous pouvons calculer (Table 2) la distribution théorique (si H_0 est vrai) :

$$e_{tt'}^{12} = \frac{n_t^1 * n_{t'}^2}{n}$$

La valeur du test correspond à la somme des carrés des différences entre les effectifs observés et théoriques divisée par l'effectif théorique :

$$X^2 = \sum_{t,t'} \frac{(n_{tt'}^{12} - e_{tt'}^{12})^2}{e_{tt'}^{12}}$$

Ici, $X^2 = 4,317$.

La valeur du test est ensuite comparée à la valeur de la loi Chi-deux $\chi_{df,\alpha}^2$ pour savoir l'hypothèse H_0 peut être rejetée avec une marge d'erreur de $\alpha\%$. Le nombre de degrés de liberté df est le produit des nombres de classes moins 1 : $df = (c^1 - 1)(c^2 - 1)$. Ici $df=1$. Avec une probabilité d'erreur de 5%, la valeur de la loi Chi-deux est $\chi_{1,0,95}^2 = 3,84$. L'hypothèse "Le sexe du candidat n'a aucun impact sur

les chances d'être recruté" peut donc être rejetée avec moins de 5% de chance de se tromper. Nous ne pouvons toutefois pas rejeter l'hypothèse avec moins d'1% de chance de nous tromper, car $\chi_{1,0,99}^2 = 6,63$.

Un des avantages de ce test est l'absence d'hypothèses sur la forme de la distribution des variables étudiées. De plus, pour être valide, le test n'a que deux conditions : l'indépendance des observations et un échantillon suffisamment large. Le critère quantitatif généralement utilisé (voir [15]) est que l'effectif théorique doit être supérieur à 1 pour chaque élément du tableau de contingence ($e_{tt'}^{ij} > 1$) et supérieur à 5 ($e_{tt'}^{ij} > 5$) dans 80% des cas. Nous utilisons cette condition, appliquée à chaque couple de variables, comme critère d'arrêt de notre cycle génération/analyse de nouvelles simulations.

3.4 Méthodologie

Dans cette partie, nous allons décrire plus en détail notre modèle, ce qui correspond à la méthodologie d'analyse suivie par notre "stagiaire artificiel".

L'algorithme principal (Alg. 1) suit les étapes représentées Fig. 1 :

```

ExtractVariables();
ChooseObjective();
ChooseConfig();
ResValid=false;
while ResValid=false do
    GenerateRunSimulation() → ResFile;
    UpdateResults(ResFile);
    UpdateClasses();
end
ShowResults();

```

Algorithm 1: Algorithme principal de l'outil d'analyse

ExtractVariables() est principalement une procédure de traitement de fichiers textes. A partir d'un fichier de paramètres (fichier utilisé pour indiquer à RePast un ensemble de simulations à réaliser) et d'un fichier résultat (contenant les résultats de ces simulations), la procédure extrait l'ensemble des noms de paramètres $v^1 \dots v^p$ (ainsi que des valeurs par défaut à proposer à l'utilisateur) et les noms des variables observées $v^{p+1} \dots v^{p+r}$.

ChooseObjective() permet à l'utilisateur de sélectionner les variables qui vont être analysées

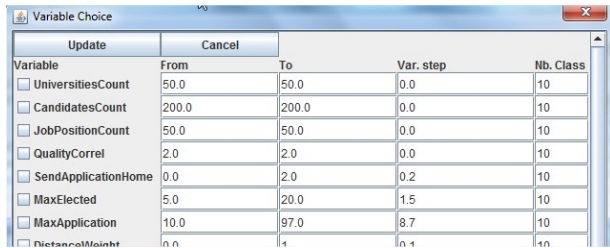


FIGURE 2 – Fenêtre de choix des variables analysées : pour chaque variable, l'utilisateur choisit s'il veut qu'elle soit utilisée pour les tests d'indépendance, le nombre de classes à utiliser et les bornes inférieures et supérieures.

(U) et, éventuellement, de définir manuellement le nombre de classes utilisées ($c^1 \dots c^{p+r}$) et leurs bornes initiales (voir Fig. 2). Par défaut, la première série de simulations est utilisée pour déterminer les bornes de $c_{default}$ classes uniformes pour chaque variable. Les classes par défaut sont uniformes en termes de taille plutôt qu'en terme de population pour plusieurs raisons : Tout d'abord, l'échantillon disponible après la première série de simulations étant assez réduit, la forme de la distribution a de grandes chances d'évoluer. Par ailleurs, il est très courant d'avoir des variables pour lesquelles les observations sont concentrées autour d'un petit nombre de valeurs (voire d'une valeur unique). Par exemple, dans notre application, 75% des observations de la variable "TotalHire", correspondant au taux de recrutement, sont concentrées sur la valeur 100%. Dans ce cas, le choix de classes homogènes en terme d'effectif pour une variable v conduirait à de très petites classes (en terme de taille) autour d'une valeur x , et à des classes plus étalées pour les autres valeurs. Cela influencerait le résultat du test d'indépendance, car une variable qui influencerait v pour des valeurs proches de x serait considérée comme ayant plus d'influence que des variables qui influencent v pour d'autres valeurs (car de plus grands changements de v seraient nécessaires pour passer d'une classe à une autre). Pour une discussion plus approfondie sur la définition manuelle des classes, voir section 3.6.

ChooseConfig() permet à l'utilisateur de choisir quelles valeurs utiliser pour les paramètres lors des simulations (voir Fig. 3). Les fichiers de paramètres de RePast acceptent les ensembles de valeurs (set) et les boucles (For..step..until). Afin que les tests soient valides, les simulations doivent être indépendantes. Ce ne serait pas le cas avec des boucles. Pour éviter ce problème, la procédure remplace automatiquement

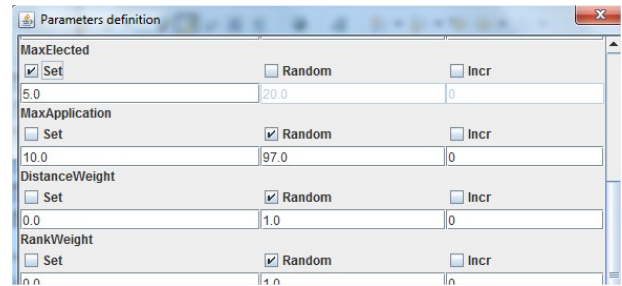


FIGURE 3 – Fenêtre de choix des paramètres de la simulation : pour chacun, l'utilisateur choisit entre un ensemble de valeur (set), une variable aléatoire continue (random) ou une variable aléatoire discrète avec écart fixe (incr).

les boucles par des choix aléatoires parmi l'ensemble des valeurs atteignables dans la boucle. Pour chaque paramètre, l'utilisateur peut alors choisir entre un ensemble de valeurs (pouvant se réduire à une valeur, set) et une valeur aléatoire dans un intervalle (continu, ou discret si c'est une boucle). Une discussion plus approfondie sur le choix des paramètres peut être trouvée en section 3.6.

GenerateRunSimulation() génère un nouveau fichier de configuration en utilisant les options choisies par l'utilisateur pour une nouvelle série de n_{brun} simulations, crée le fichier de script adapté et lance la simulation. Pour accélérer la validation des tests, les valeurs aléatoires d'un des paramètres peuvent être biaisées lors de cette phase (voir section 3.5).

UpdateRes(ResFile) est décrit Alg. 2. Dans la boucle *While*, la procédure met à jour le tableau de contingence correspondant à la population observée (les $n_{tt'}^{ij}$). Dans la boucle *For*, elle calcule les tests de Chi-deux et vérifie s'ils sont tous valides pour déterminer si le critère d'arrêt (*ResValid*) est atteint. La condition de validité (voir section 3.3) est que les effectifs théoriques ($e_{tt'}^{ij}$) soient tous supérieurs à 1 et que 80% d'entre eux soient supérieurs à 5, et ce pour chaque couple de variables testé. La valeur $P_{r^{ij}}$ calculée correspond à la marge d'erreur minimale selon laquelle H_0 (v^i et v^j sont indépendants) peut être rejetée. Cette valeur donne une information plus riche qu'une simple réponse binaire rejetée/non rejetée avec un seuil, et permet à l'utilisateur de comparer les différentes valeurs entre elles.

UpdateClasses() est utilisée pour ajouter/fusionner des classes. Une classe est ajoutée si une valeur observée est inférieure à la borne

inférieure de la première classe (ou supérieure à la borne supérieure de la dernière), à la condition que l'utilisateur n'ait pas choisi que ces bornes soient infinies ou d'interdire la création de classe. L'utilisateur peut en effet préférer conserver uniquement les valeurs comprises entre les bornes extrêmes (et ignorer les autres), en particulier lorsque les bornes sont définies manuellement, ou encore vouloir considérer les deux classes extrêmes comme infinies. La fusion de classes est une heuristique permettant d'accélérer la validation des conditions d'arrêt et d'éviter certains cas de blocage (voir section 3.5).

```

while new res  $x^1 \dots x^{p+r}$  in ResFile do
  inc(n.);
  for each  $v^i, v^j$  with  $v^i \in U$  and  $v^j \in U$  and  $i \leq j$  do
    for each class  $c_t^i$  do
      if  $l_t^i \leq x^i < l_{t+1}^i$  then inc( $n_t^i$ )
    end
    for each  $c_t^i \times c_{t'}^j$  do
      if  $l_t^i \leq x^i < l_{t+1}^i$  and  $l_{t'}^j \leq x^j < l_{t'+1}^j$ 
        then inc( $n_{tt'}^{ij}$ )
      end
    end
  end
end
ResValid = true;
for each  $v^i, v^j$  with  $v^i \in U$  and  $v^j \in U$  and  $i \leq j$  do
  Nb5 = 0;
  for each  $c_t^i \times c_{t'}^j$  do
     $e_{tt'}^{ij} = \frac{n_t^i * n_{t'}^j}{n}$ ;
    if  $e_{tt'}^{ij} < 1$  then ResValid = false;
    if  $e_{tt'}^{ij} < 5$  then inc(Nb5);
  end
   $X^{ij} = \sum_{t,t'} \frac{(n_{tt'}^{ij} - e_{tt'}^{ij})^2}{e_{tt'}^{ij}}$ ;
   $Pr^{ij} = pr$  with  $\chi_{(c^i-1)(c^j-1),pr}^2 = X^{ij}$ ;
  if Nb5 > 0.2 $c^i c^j$  then ResValid = false
end

```

Algorithm 2: UpdateRes(ResFile)

ShowResults() présente à l'utilisateur les résultats obtenus (les Pr^{ij} , pouvant être interprétés comme la probabilité pour v^i et v^j de ne pas être indépendants). L'utilisateur peut avoir le détail des populations observées/théoriques pour chaque couple de variables en sélectionnant la valeur correspondante (voir des exemple Fig. 4 et Fig. 5).

3.5 Heuristiques de convergence

Le temps d'exécution non négligeable de certains types de simulations, et le fait que plusieurs dizaines de milliers de simulations puissent être nécessaires pour que tous les tests d'indépendance soient valides, rend l'utilisation d'heuristiques pour accélérer la convergence vers des tests valides potentiellement intéressante. Nous proposons deux heuristiques, pouvant être utilisées simultanément :

La **fusion de classes** lors de la phase UpdateClasses (voir section 3.4) : une classe est fusionnée avec son voisin contenant la plus petite population si son effectif est inférieur à $minclass\%$ l'effectif moyen de la variable. Cette heuristique permet d'éviter qu'une classe existante à cause d'une valeur aberrante voire inatteignable (par exemple une classe sur l'intervalle [1,2 ; 1,8] pour une variable ne prenant que des valeurs entières) ne bloque les conditions de terminaison (les valeurs théoriques associées à cette classe seraient alors toujours inférieures à 1). Elle permet de plus d'accélérer la convergence car ce sont les classes avec la plus faible population qui ne remplissent pas les conditions de validité. L'avantage de cette heuristique est qu'elle est particulièrement efficace (voir section 5). La principale limite de cette heuristique est qu'elle est inapplicable dans le cas de variables non ordonnées (car il est impossible de savoir quelles classes fusionner) ou lorsque les classes sont fixes (lorsque l'utilisateur veut analyser des classes précises).

Le **choix biaisé des valeurs aléatoires** pour un paramètre lors de la procédure GenerateRunSimulation() présente l'avantage de pouvoir s'appliquer avec n'importe quel type de variable. Le principe est d'identifier la variable et sa classe qui pose le plus problème, identifier le paramètre permettant d'augmenter avec le plus de chances l'effectif de cette classe, et de biaiser les valeurs de ce paramètre. Plus formellement

- Choix de la classe t et de la variable i t.q. $n_t^i = \min_{j,t'} n_{t'}^j$
- Pour chaque paramètre aléatoire continu v^j , le score de chaque classe est de $s_{t'}^j = \left(\frac{n_{tt'}^{ij}}{\sum_t n_{tt'}^{ij}} \right)^2$
- Pour la série de simulations suivante, le paramètre biaisé v^j est celui qui a le score total $s^j = \sum_{t'} s_{t'}^j$ le plus élevé.
- A chaque tirage aléatoire, une classe t' de v_j est choisie avec une probabilité proportion-

nelle à $s_{t'}^j$. La valeur du tirage est choisie de façon uniforme dans la classe t' .

Cette heuristique n'est applicable qu'à un paramètre par série de simulations, car si le biais était appliqué à plusieurs paramètres simultanément, cela introduirait un lien artificiel qui pourrait influencer les tests d'indépendance.

3.6 Discussion

L'objectif de notre méthode est de tester l'indépendance des différentes variables d'un modèle. Toutefois, l'outil obtenu peut être utilisé de plusieurs façons :

- La première utilisation consiste à donner un premier aperçu global d'un modèle de simulation. Un modèle peut avoir plusieurs centaines de paramètres/variables observées, et obtenir une vision globale des liens entre toutes ces variables est à la fois utile et complexe à réaliser avec les plateformes actuelles de simulation. C'est l'objectif par défaut de notre modèle, avec tous les paramètres par défaut et toutes les variables sélectionnées pour l'analyse (voir un exemple de résultat suivant cet objectif en section 5.1).
- Un second objectif peut être d'observer le comportement d'un modèle pour un sous-ensemble de valeurs des paramètres, par exemple autour d'un point d'équilibre. Définir un sous-modèle avec des paramètres fixés ou limités peut avoir des répercussions importantes sur les résultats obtenus (voir en section 5.2 pour un exemple).
- Un troisième objectif consiste à étudier le niveau d'influence d'une variable sur une autre en changeant le nombre de classes. Avec un faible nombre de classes pour une variable v^1 , seules les variables ayant une grande influence sur v^1 pourront influencer sur la classe résultat, et seront donc considérées comme non indépendantes (voir un exemple en section 5.3).

4 Exemple d'application : le marché des maîtres de conférences

Pour illustrer notre modèle et l'outil associé, nous l'avons appliqué à une simulation du processus de recrutement des maîtres de conférences. Le seul objectif de cette application étant d'illustrer notre approche, une compréhension détaillée du modèle de simulation n'est pas nécessaire et nous nous contenterons de le décrire brièvement (voir [4, 5] pour plus de détails).

Soit $\{u_1, \dots, u_U\}$ et $\{c_1, \dots, c_C\}$ respectivement l'ensemble des universités et des candidats, ordonnés selon leur qualité. Une université u_t est caractérisée par trois paramètres. Les deux premiers (définis dans $[0..1]$) définissent sa fonction de préférence : i) son élitisme e_t mesure sa préférence envers les meilleurs candidats ; ii) son localisme l_t mesure sa préférence envers les candidats locaux. Les autres critères de préférence sont modélisés par une variable aléatoire V avec un poids $(1 - e_t)$. La préférence d'une université envers un candidat est de :

$$r(i, t) = (r_t \times \frac{i}{C} + (1 - e_t)V)(1 - l_t.L(i, t))$$

$L(i, t)=1$ ssi c_i est local pour u_t et 0 sinon. Pour choisir les candidats auditionnés et classés, chaque université dispose également d'un facteur r_t d'aversion au risque la conduisant à préférer les candidats avec un rang proche du sien. Elle va classer un candidat i ayant candidaté chez elle selon sa fonction d'évaluation :

$$s(i, t) = r_t \times r(i, t) + (1 - r_t) \times \frac{|i - t|}{C}$$

Les paramètres des candidats, leur fonction de préférence $r'(i, t)$ et d'évaluation $s'(i, t)$ sont similaires à ceux des universités.

Règles d'interaction Chaque candidat c_i postule dans $NbApp_i$ universités en suivant sa fonction d'évaluation, ainsi que dans son université d'origine. Chaque université u_t classe 5 candidats en suivant sa fonction d'évaluation. Chaque candidat classe les universités qui l'ont lui-même classé selon sa fonction de préférence. Tous ces classements sont centralisés par le ministère, qui réalise l'affectation finale à l'aide d'un algorithme de mariage stable optimal pour les candidats ([3]).

Les valeurs des paramètres des différents agents sont définies à l'aide de variables aléatoires dont les bornes sont les paramètres de la simulation. Par exemple :

- $NbApp_i \sim U[MinApp, MaxApplication]$,
- $r_t \sim U[MinUnivRisk, MaxUnivRisk]$

Les variables observées sont le nombre de recrutements ($NbHire$), le nombre de recrutements locaux ($NbLocalHire$), le rang du dernier candidat recruté ($LastRelACand$) et le rang du premier poste non pourvu ($FirstRelNAJob$).

Population										
Nb. XP										
4182	SendApplication	HomeMaxApplication	DistanceWeight	RankWeight	MinCandRisk	MaxCandRisk	MinUnivRisk			
LastRelACand	0.52819	*1.0	*1.0	*0.88911	0.94453	0.16809	0.86364			
FirstRelNAJobs	0.86139	*1.0	*1.0	*1.0	0.19615	0.22345	0.84257			
NbLocalHire	*1.0	*1.0	*0.98337	*1.0	0.66491	0.5987	0.31528			
NbHire	0.93636	*1.0	0.07526	0.1797	0.45918	0.31576	*0.97833			
MaxLocalBonus	0.86105	0.20626	0.2597	0.46154	0.07444	0.34659	0.37135			
MinLocalBonus	0.16485	0.78863	0.3636	0.66905	0.84726	0.40142	0.90696			
UnivRankWeight	0.08951	0.51596	0.44967	0.20327	0.83975	0.44966	0.30117			
MaxUnivRisk	0.34731	0.57991	0.22055	0.15469	0.81108	0.78509	0.70703			
MinUnivRisk	0.8062	0.71874	0.34139	0.89567	0.67724	0.94328	*1.0			
MaxCandRisk	0.22829	0.6967	0.117	0.1204	0.18872					
MinCandRisk	0.03377	0.18427	0.59667	0.71646	*1.0					
RankWeight	0.75229	0.89995	0.29689	*1.0						

FIGURE 4 – Fenêtre de résultat : chaque cellule correspond à la probabilité pour un couple de variables de ne pas être indépendantes. Les étoiles indiquent les couples pour lesquels la valeur est supérieure à 0,95.

Detail						
Nb. XP						
Diff sum 1402.958591343494						
DL 99						
Proba indep. 0.0						
Proba not indep. 1.0						
NbHire	MaxApplication	10.0>	18.7>	27.4>	36.1>	44.8>
4182	4182	420	422	413	395	455
0.15>	93	0.0/0.52683	0.0/0.5722	0.0/0.36805	0.0/0.95976	7.0/10.3
0.19>	61	0.0/6.24878	0.0/6.27854	0.0/6.14463	1.0/5.87683	4.0/6.76
0.21>	175	0.0/17.92683	0.0/18.0122	2.0/17.62805	10.0/16.85976	23.0/19.
0.23>	239	0.0/24.48293	0.0/24.59951	5.0/24.07488	14.0/23.02561	18.0/26.
0.26>	328	0.0/33.6	3.0/33.76	10.0/33.04	16.0/31.6	45.0/36.
0.29>	353	0.0/36.16098	4.0/36.33317	12.0/35.55829	38.0/34.00854	64.0/39.
0.32>	303	0.0/31.03902	8.0/31.18683	40.0/30.52171	43.0/29.19146	49.0/33.
0.35>	288	3.0/29.50244	27.0/29.64293	39.0/29.01073	51.0/27.74634	35.0/31.
0.38>	302	8.0/30.93659	50.0/31.0839	62.0/30.42098	33.0/29.09512	32.0/33.
0.41>	311	26.0/31.85854	53.0/32.01024	51.0/31.32756	36.0/29.9622	32.0/34.
0.44>	454	84.0/46.50732	74.0/46.72878	51.0/45.7322	42.0/43.73902	47.0/50.
0.47>	1193	299.0/122.20976	203.0/122.79171	141.0/120.17293	111.0/114.93537	99.0/13.

FIGURE 5 – Détail des données du test d'indépendance entre *MaxApplication* et *NbHire*. Les en-têtes de lignes et colonnes contiennent les bornes des classes. Chaque cellule contient les effectifs observés et théoriques.

5 Résultats

Dans un premier temps, nous allons présenter des exemples d'utilisation de notre outil pour chaque cas d'application : pour obtenir une vision d'ensemble d'une simulation (5.1), pour étudier le modèle autour d'un point précis (5.2), et pour étudier l'importance des liens en changeant le nombre de classes (5.3). Enfin, nous allons tester l'efficacité des différentes heuristiques de convergence (5.4).

5.1 Vision d'ensemble

Avec les paramètres par défaut et en sélectionnant toutes les variables pour l'analyse, l'outil fournit une vision d'ensemble des liens du modèle (Fig. 4). En haut de la fenêtre est indiqué le nombre de simulations réalisées afin que tous les tests soient valides (4182)³. Chaque ligne et chaque colonne correspond à une variable (les 4 premières lignes correspondant aux

3. Réaliser l'analyse (comprenant les 4182 simulations avec 250 agents) prend - pour notre application - environ 6 minutes.

variables observées et les suivantes aux paramètres de la simulation). Chaque cellule du tableau contient le résultat du test ($P_{r^{ij}}$). Un test positif avec moins de 5% de chance de se tromper ($P_{r^{ij}} > 0,95$) est signalé par une étoile. Cela signifie que l'hypothèse selon laquelle ces deux variables sont indépendantes devrait être rejetée (avec moins de 5% de marge d'erreur). La fenêtre fournit à l'utilisateur une vision d'ensemble du fonctionnement de son modèle de simulation : le paramètre *MaxApplication*, déterminant le nombre de dossiers envoyés par le candidat, est clairement celui qui a le plus d'influence sur les variables observées. On peut noter que c'était un des résultats de l'étude originale ([4]), mais sans validation statistique, et avec comme seul argument un graphique et une interprétation. Avec cette première fenêtre de résultat, il est possible d'affirmer que c'est le seul paramètre pour lequel l'hypothèse d'indépendance peut être rejetée avec la plus faible marge d'erreur (10^{-5}) pour chacune des variables observées. D'autres résultats peuvent être confirmés statistiquement : par exemple, le fait que les candidats soit risque-takers ou risque-averses (*MinCandRisk* et *MaxCandRisk*) a très peu d'influence sur les variables observées. De plus, en sélectionnant une des cellules, l'utilisateur peut obtenir le détail des effectifs observés et théoriques. Par exemple, la Fig. 5 donne le détail pour le couple *MaxApplication* (nombre de dossiers maximum envoyés par candidat) et *NbHire* (nombre de candidats recrutés pour 50 postes). Il apparaît alors que le paramètre a un impact négatif sur la variable observée. On peut enfin remarquer que pour les couples paramètre/paramètre (toutes les valeurs fig. 4 à partir de la cinquième ligne) les probabilités ne sont pas toutes faibles. Les valeurs des paramètres suivent pourtant toutes des variables aléatoires continues (utilisant la fonction Random de Java, et non biaisées, l'heuristique de biais n'est introduite qu'en section 5.4). Pourtant, pour certains couples, l'hypothèse d'indépendance pourrait être rejetée à condition de considérer des marges d'erreur suffisamment élevées (15%). Cela montre qu'il est important de considérer des marges d'erreur assez faibles (5%, 1% ou moins) pour pouvoir affirmer que deux variables ne sont pas indépendantes.

5.2 Influence du choix des paramètres

Lors de l'analyse précédente, la totalité de l'espace des paramètres était explorée. Il peut être intéressant d'étudier l'influence des va-

	Nb.XP	MaxApplication	Distance	Weight	RankWeight	MinCandRisk	MaxCandRisk	MinUnivRisk
3468								
LastRelACand	0.91423	0.7009		*1.0	*1.0	*1.0	0.70544	
FirstRelNAJobs	*0.99992	0.79006		0.82513	*1.0	0.84114	0.79328	
NBLocalHire	0.8412	*1.0		0.84399	*0.98821	0.67382	0.23346	
NBHire	*0.9991	*0.99963		*1.0	*1.0	0.20851	0.90913	

FIGURE 6 – Fenêtre de résultat avec des valeurs de paramètres limitées autour d’un point.

riables autour d’un point donné, ou avec certains paramètres fixes. Pour étudier le comportement de notre simulation autour de l’équilibre observé empiriquement ([4]), nous avons fixé certains paramètres (par exemple *SendApplicationHome*, qui n’apparaît plus dans l’analyse), et limité les valeur des autres à +/- 10% autour de l’équilibre. Les résultats sont présentés Fig. 6. Dans cette configuration, les liens entre les variables ont changé. L’influence de *MaxApplication*, dont l’espace des valeurs est passé de [10..97] à [18..22], est par exemple très différente : les valeurs élevées du paramètre, qui provoquaient une saturation du système, ont disparu. Même sans ce phénomène de saturation, une influence sur certaines variables (*FirstRelNAJob* et *NbHire*) demeure. Par ailleurs, certaines influences, auparavant trop faibles pour être mesurées, peuvent apparaître : l’influence de *MinCandRisk*, trop faible pour une analyse globale, est suffisante pour être mesurée autour de ce point spécifique.

5.3 Influence du choix des classes

Il est également possible de modifier le nombre de classes pour certaines variables ou de définir manuellement ces classes pour l’analyse. Diminuer le nombre de classes diminue la sensibilité de l’analyse, mais ce “floutage” peut permettre d’identifier les liens les plus importants. Par exemple, la Fig. 7 donne le détail de l’analyse pour le paramètre *RankWeight* (élitisme des universités) et la variable *NbLocalHire* (nombre de recrutements locaux) avec deux configurations : une avec initialement 10 classes (figure du haut), l’autre avec initialement 3 classes (figure du bas). Le nombre de classes final est différent car l’heuristique de fusion de classes est utilisée. Dans la première expérience, même si le test indique que les variables ne sont pas indépendantes (erreur inférieure à 0,1%), l’influence n’apparaît pas clairement lorsqu’on analyse les populations observées et théoriques (contrairement par exemple à celle de la Fig. 5). Diminuer la précision en diminuant le nombre de classes dans la deuxième expérience fait alors

	Nb.XP	Dif sum	DL	Proba indep.	Proba not indep.
10 classes	142.49730729647922	45	4.6467244726613E-12	0.999999999995333	0.000000000004667
3 classes	2.62228669934629	2	0.26951173400514516	0.7304882659948548	0.26951173400514516

NBLocalHire	RankWeight	0.0>	0.19>	0.38>	0.57>	0.76>	0.95>
4182	219	482	428	441	446	446	228
0.0>	1839	67.0/98.22951	186.0/216.19463	172.0/191.07859	166.0/197.80463	176.0/200.04732	112.0/112.0
0.046>	1040	73.0/55.55122	136.0/122.26341	122.0/108.05854	125.0/111.86341	130.0/113.13171	53.0/57.0
0.092>	451	38.0/24.09	69.0/53.02	51.0/46.86	51.0/48.51	54.0/49.06	27.0/25.0
0.138>	305	21.0/16.29146	47.0/35.8561	34.0/31.69024	28.0/32.8061	26.0/33.17805	18.0/16.0
0.184>	287	10.0/15.33	21.0/33.74	38.0/29.82	43.0/38.87	36.0/31.22	9.0/15.9
0.276>	178	10.0/9.5078	23.0/20.92585	9.0/18.49463	28.0/19.14585	24.0/19.36293	9.0/8.89

FIGURE 7 – Détail des tests d’indépendance pour le paramètre *RankWeight* et la variable observée *NbLocalHire*. Le nombre initial de classes par variable est fixé à 10 dans l’expérience 1 (haut), et à 3 dans l’expérience 2 (bas).

TABLE 3 – Nombre de simulations moyen (et écart-type) avec/sans l’heuristique de fusion de classe (HC) et l’heuristique de simulation biaisée (HB)

	HC	\overline{HC}
$\frac{HB}{HB}$	4760(1517)	16442(1943)
$\frac{HB}{HB}$	6800(1819)	24112(579)

disparaître ce lien faible de l’analyse et l’hypothèse d’indépendance ne peut plus être rejetée.

5.4 Test des heuristiques

Pour tester l’efficacité des heuristiques présentées section 3.5, nous avons analysé plusieurs fois la même configuration en activant/désactivant chaque heuristique et nous avons mesuré le nombre de simulations avant que les conditions de validité soient vérifiées. Cette configuration comportait 10 classes par variable, certaines classes étant particulièrement difficiles à atteindre par la simulation. Les résultats sont présentés Tab. 3⁴. L’heuristique *HC* (fusion de classe) apparaît comme la plus efficace et diminue significativement le nombre de simulations à réaliser. Dans le cas de variables pour lesquelles cette heuristique est inapplicable (texte, ...) ou si l’utilisateur souhaite des classes fixes, l’heuristique *HB* permet d’obtenir un gain de temps, certes plus réduit, mais sans contrainte particulière.

4. Les différences entre les heuristiques sont significatives (selon un test du Chi-deux bien sûr !) avec une marge d’erreur inférieure à 1%.

6 Conclusion

Dans cet article, nous avons présenté une méthode de conception d'un outil permettant de générer automatiquement de nouvelles simulations, d'analyser et de présenter des résultats statistiquement valides d'indépendances entre variables. Des heuristiques ont de plus été proposées et testées pour réduire le nombre de simulations nécessaires. Nous avons présenté un outil correspondant, générique pour toute simulation de la plateforme RePast, et l'avons appliqué à une simulation de marché du travail académique. Nous avons montré que le modèle permettait d'obtenir à la fois une vision d'ensemble d'un modèle de simulation, mais également des analyses plus fines ou plus détaillées. Ces résultats peuvent être obtenus avec un minimum d'intervention de l'utilisateur ou de connaissances des méthodes statistiques sous-jacentes, l'outil remplissant ainsi son rôle de "stagiaire artificiel". Les principales perspectives d'évolution concernent la généralisation à d'autres plateformes et à d'autres tests statistiques. Parallèlement, une intégration en plugin d'une plateforme spécifique, ou au projet SimExplorer, très complémentaire, est envisagée.

Références

- [1] Frédéric Amblard, David R. C. Hill, Stéphane Bernard, Jérôme Truffot, and Guillaume Deffuant. <http://www.simexplorer.org/>.
- [2] R. Axelrod, 'Advancing the art of simulation in the social sciences', *Advances in Complex Systems*, **7**(1), 77–92, (2004).
- [3] Mourad Baiou and Michel Balinski, 'Student admissions and faculty recruitment', *Theor. Comput. Sci.*, **322**(2), 245–265, (2004).
- [4] Philippe Caillou and Michèle Sebag, 'Modelling a Centralized Academic Labour Market : Efficiency and Fairness', in *ECCS08*, Jerusalem Israel, (2008). Complex Systems Society.
- [5] Philippe Caillou and Michèle Sebag, 'Pride and Prejudice on a Centralized Academic Labor Market', in *Artificial Economics 09*, LNEMS, pp. 29–40, Valladolid Espagne, (2009). Springer-Verlag.
- [6] P. E. Greenwood and M. S. Nikulin, *A Guide to Chi-Squared Testing*, Wiley, New York, NY, 1996.
- [7] Brian Heath, Raymond Hill, and Frank Ciarallo, 'A survey of agent-based modeling practices (january 1998 to july 2008)', *Journal of Artificial Societies and Social Simulation*, **12**(4), 9, (2009).
- [8] Scott Moss, 'Alternative approaches to the empirical validation of agent-based models', *Journal of Artificial Societies and Social Simulation*, **11**, (2007).
- [9] M.J. North, N.T. Collier, and J.R. Vos, 'Experiences creating three implementations of the repast agent modeling toolkit', *ACM Transactions on Modeling and Computer Simulation*, **16**(1), 1–25, (2006).
- [10] D. Phan, 'From agent-based computational economics towards cognitive economics', in *Cognitive Economics*, Handbook of Computational Economics, 371–398, Springer Verlag, (2004).
- [11] Denis Phan and Frederic Amblard, *Multi-agent Modelling and Simulation in the Social and Human Sciences*, Bardwell Press, <http://www.bardwell-press.co.uk/>, septembre 2007.
- [12] Steven F. Railsback, Steven L. Lytinen, and Stephen K. Jackson, 'Agent-based simulation platforms : Review and development recommendations', *Simulation*, **82**(9), 609–623, (2006).
- [13] L. Soldatova, A. Clare, A. Sparkes, and R. D. King, 'An ontology for a robot scientist', *Bioinformatics*, **22**, 464–471, (2006). ISMB06.
- [14] Leigh S. Tesfatsion, 'A constructive approach to economic theory', in *Handbook of Computational Economics*, volume 2 Agent-Based Computational Economics of *Handbooks in Economic Series*, North-Holland, (2006).
- [15] Cochran WG, 'Some methods for strengthening the common chi-square tests', *Biometrics*, (10), 10–417, (1954).
- [16] Uri Wilensky. <http://ccl.northwestern.edu/netlogo/>.
- [17] Paul Windrum, Giorgio Fagiolo, and Alessio Moneta, 'Empirical validation of agent-based models : Alternatives and prospects', *Journal of Artificial Societies and Social Simulation*, **10**, (2007).
- [18] Thomas H. Wonnacott and Ronald J. Wonnacott, *Introductory statistics [by] Thomas H. Wonnacott [and] Ronald J. Wonnacott*, Wiley New York., 5th ed. edn., 1990.