Réflexions pour une approche reproductible des travaux de recherche sur les systèmes multi-agents

Axel Carayon, Frédéric Migeon

Réflexions pour une approche reproductible des travaux de recherche sur les systèmes multi-agents

Axel Carayon axel.carayon@irit.fr
Frederic Migeon frederic.migeon@irit.fr

IRIT - Institut de Recherche en Informatique de Toulouse
Université de Toulouse, CNRS, Toulouse INP, UT3, UT1, Toulouse, France

Abstract

Les systèmes multi-agents sont un paradigme de programmation dont l'objectif est d'obtenir un système logiciel sous la forme d'un système complexe composés d'agents en interaction. L'objectif est de retrouver les propriétés d'émergence des systèmes complexes pour obtenir la fonctionnalité attendue alors que ces systèmes sont plongés dans des environnements déterministes non prédictifs. Les travaux de recherche sur ces systèmes multi-agents imposent d'avoir une démarche empirique permettant de comprendre et de valider les résultats. Afin d'améliorer la qualité des résultats produits lors des expérimentations de ces systèmes et des avancées scientifiques sur les développements de systèmes complexes numériques, nous travaillons sur un prototype de démarche reproductible impliquant les systèmes multi-agents. Cet article présente un état de l'art sur la reproductibilité et en rapproche certaines activités des méthodes de développement logiciel. En regard des objectifs de répétabilité, réplicabilité et reproductibilité attendus, nous lançons la discussion sur une telle approche à partir d'un premier prototype de démarche scientifique.

1 Introduction

Pour la publication scientifique, la validation par les pairs est un élément clef pour la validation des résultats d'un travail de recherche et de ses expérimentations. Dans la volonté de pouvoir offrir ce type de validation lors des expérimentations, de faciliter la comparaison des résultats et d'améliorer leurs publications, nous nous sommes penchés sur la question de la reproductibilité qui impacte le processus de lecture, le travail collectif et la qualité intrinsèque de la publication [23] pour laquelle une plus grande confiance offre une plus-value aux travaux [14].

La question de la reproductibilité se pose dans tous les domaines de recherche. À l'heure actuelle, les publications scientifiques n'offrent pas beaucoup de place à l'aspect reproductible des expérimentations qu'elles comportent, tant dans le manque d'informations fournies que dans le manque de documentation des méthodes adoptées. Depuis quelques années, plusieurs scientifiques tirent la sonnette d'alarme et alertent sur une *crise de la reproductibilité* [15][3] en recherche. Dans leur état de l'art, Odd Erik Gundersen et al [13] mettent en évidence cette crise de la reproductibilité que traverse actuellement le domaine de l'intelligence artificielle. Sur 400 participations aux conférences IJCAI et AAAI, à peine 6% des auteurs partageaient leur code et 30% partageaient leur jeu de données de test. Cela montre à quel point la pratique de l'évaluation approfondie de la validation des résultats est déficiente.

Il existe plusieurs barrières à la production de publications de travaux reproductibles : les barrieres institutionnelle, juridique et technique. Parmi les facteurs importants, on retiendra le peu de valorisation accordé à des publications de travaux reproductibles en regarde de l'effort nécessaire [22], la privatisation des données ou la propriété intellectuelle, les accès payants à certaines publications, le manque de précision des protocoles expérimentaux [10], des environnements de déploiement, des codes source et parfois des données [7]. A cela, il ne faut pas soustraire la question très pragmatique des artefacts nécessaires et suffisants à publier pour permettre la compréhension des méthodes et leur application à d'autres domaines [24].

Face à ce constat, les travaux de recherche sur les systèmes multi-agents semblent plus proches des sciences computationnelles [1] que des sciences mathématiques quant à la validation de leurs résultats. Il est probable que les caractéristiques des systèmes multi-agents qui en font des sys-

1. https://en.wikipedia.org/wiki/Computational_science

tèmes complexes numériques en soient la cause.

Cet article propose de lancer le débat de la reproductibilité des travaux de recherche sur les systèmes multi-agents et, en particulier, au sein de la communauté des JFSMA, et propose quelques pistes méthodologiques pour améliorer cette caractéristique.

Après avoir rappelé les caractéristiques des systèmes multi-agents et des systèmes complexes, la section 2 présente quelques caractéristiques de l'approche empirique nécessaires à la validation de ces systèmes. La section 3 propose un état de l'art de la reproductibilité pour en extraire quelques définitions et réflexions méthodologiques. Cette section sera complétée par une présentation des pratiques consensuelles en développement logiciel qui pourrait participer à une meilleure reproductibilité. A travers un exemple, la section 4 propose quelques étapes concrètes et lance la discussion sur les pratiques à introduire pour la reproductibilité de nos travaux. La dernière section présente quelques éléments de conclusion et de perspectives.

2 Systèmes complexes et systèmes multi-agents

Appréciés en ingénierie logicielle notamment pour les qualités qu'ils apportent en architecture logicielle [2], les systèmes multi-agents sont un objet d'étude et un outil de recherche pour lequel les protocoles d'expérimentation nécessitent d'être précisés. Comparable à des systèmes complexes dans un environnement numérique, les systèmes multi-agents nécessitent donc une approche empirique pour la validation des résultats. Cette section décrit les caractéristiques et propriétés des systèmes complexes sur lesquelles nous nous focalisons. Bien que la notion de système multi-agent n'ait plus à être présentée dans la communauté des JFSMA, nous rappelons quelques définitions et propriétés qui permettent d'illustrer pourquoi ils peuvent être considérés comme des systèmes complexes numériques.

2.1 Caractérisation des systèmes complexes

Bien qu'il soit difficile de donner une définition consensuelle des systèmes complexes, nous prenons le parti de faire une présentation aussi concise que possible des concepts liés aux systèmes complexes. Le lecteur peut consulter le site *La Révolution du complexe* [2] de Janine Guespin-Michel ou l'ouvrage *Émergence, complexité et dialectique* [19] qu'elle a coordonné.

Il est souvent convenu de caractériser les systèmes complexes par un grand nombre d'entités en interaction donnant lieu parfois à des boucles de retro-action. Ces systèmes exhibent des comportements dynamiques et les phénomènes observés se décrivent comme non-linéaires ou multistationnaires. Ces systèmes font également preuve de phénomènes émergents, dont l'observation macroscopique n'est pas déductible des connaissances microscopiques du système, et de faculté d'auto-organisation, qui montre une modification de l'organisation globale sans qu'un lien évident ne puisse être déduit des modifications microscopiques du système. Enfin, caractéristique importante si l'on veut faire un parallèle avec le monde numérique, ces systèmes sont déterministes mais non prédictifs ce qui signifie que la connaissance des états précédents et présent du système ne permettent pas d'en prédire un état futur.

2.2 Définition et caractérisation des systèmes multi-agents

Selon la définition devenue classique de Ferber [11], un agent est une entité logicielle ou physique autonome, plongée dans un environnement qu'elle est capable de percevoir et sur lequel elle peut agir et dont elle possède une représentation partielle. Son cycle de vie est une boucle composée des étapes de perception, décision et action. Elle est capable de communiquer avec d'autres agents, peut posséder des ressources, des compétences et offrir des services. La fonction globale d'un système multi-agent est la résultante de l'organisation des agents qui le composent.

2.3 Les systèmes multi-agents sont des systèmes complexes numériques

Composés de nombreux agents en interaction, les systèmes multi-agents paraissent exhiber les caractéristiques décrites pour les systèmes complexes. L'émergence, propriété convoitée autant en simulation centrée individu à base d'agents qu'en résolution de problème, apprentissage ou optimisation, est obtenue par la définition des comportements des agents, au niveau micro. Pour renforcer ceci, des travaux en SMA visent à

2. http ://www.revolutionducomplexe.fr

développer les facultés d'émergence et d'auto-organisation.

La démarche scientifique qui accompagne nos recherches sur les système multi-agents est par nature empirique : elle consiste à montrer la validité à construire des systèmes complexes numériques. Soucieux d'assurer la reproductibilité de nos résultats, nous lançons la discussion d'une démarche itérative et incrémentale de reproductibilité de nos recherches.

3 État de l'art sur la reproductibilité en Informatique

Comme souvent sur des sujets interdisciplinaires, il est difficile d'isoler une définition consensuelle [12]. On peut même remarquer l'évolution de l'ACM entre les définitions sur son site web [3] et celles repérées dans [4]. Nous proposons donc ici quelques définitions et réflexions méthodologiques obtenus de l'étude bibliographique.

3.1 Définitions

Alors que les académies nationales des sciences, de l'ingénierie et de la médecine ont favorisé une définition simple de la reproductibilité comme l'obtention de "résultats cohérents en utilisant les mêmes données d'entrée, étapes computationnelles, méthodes, code et conditions d'analyse"[21], nous allons chercher la précision de plusieurs termes pour qualifier les différents degrés de reproductibilité des travaux.

Selon Tatman & al.[24], différents niveaux de reproductibilité peuvent être établis selon le niveau de portabilité du code, de sa disponibilité ainsi que de ses données.

Ince et al. [16], quant à eux, différencient la reproductibilité directe de la reproductibilité indirecte. La *reproductibilité directe* fait référence à la recompilation et à la ré-exécution du code dans un autre environnement. Par exemple, une combinaison différente de matériel et de logiciel système, afin de détecter des erreurs intrinsèques à l'expérience et non pas dues au contexte expérimental. *La reproductibilité indirecte*, pour sa part, fait référence à des efforts pour valider autre chose que le package de code entier, par exemple un sous-ensemble d'équations ou un module de code particulier.

3. https://www.acm.org/publications/policies/artifact-review-badging

Stoden & al.[22] proposent différentes bonnes pratiques sur les données, le code et l'environnement à partager. En tant que société savante, l'ACM a également proposé une définition de répétabilité, reproductibilité et de réplicabilité associées à différentes valeurs de badges garantissant l'évaluation des artefacts de l'expérience, leur disponibilité et les résultats de l'expérience.

En se basant sur les travaux Sarah Cohen-Boulakia et al. [9], nous proposons de caractériser la reproductibilité dans cet article selon les 5 éléments suivants :

- L'environnement d'exécution de l'expérience, c'est à dire le contexte dans lequel celle-ci se déroule (système d'exploitation, dépendances, librairies ...)
- L'ensemble des données utilisées par l'expérience, ainsi que la totalité des paramètres utilisés sur chacun des outils
- Le flux d'exécution de l'expérimentation, qui détaille chaque étape de son exécution et l'enchaînement de chaque action effectuée
- Les résultats de l'expérimentation ainsi que l'ensemble des traces et données qui ont été générées
- La conclusion que l'on tire de l'expérimentation

Nous proposons alors les quatre définitions suivantes, adaptées de Benureau et Rougier [6] et prenant en compte ces cinq critères illustrés par la figure 1 :

- Ré-exécutabilité : Sur la même machine, être capable de relancer l'expérience avec les mêmes données, le même processus et le même environnement.
- Répétabilité : Capacité de pouvoir ré-exécuter la même expérience pour produire le même résultat et la même conclusion.
- Réplicabilité : Avec les mêmes données et le même processus mais un environnement différent, arriver à la même conclusion.
- Reproductibilité : Être capable de produire la même conclusion scientifique quel que soit le contexte.

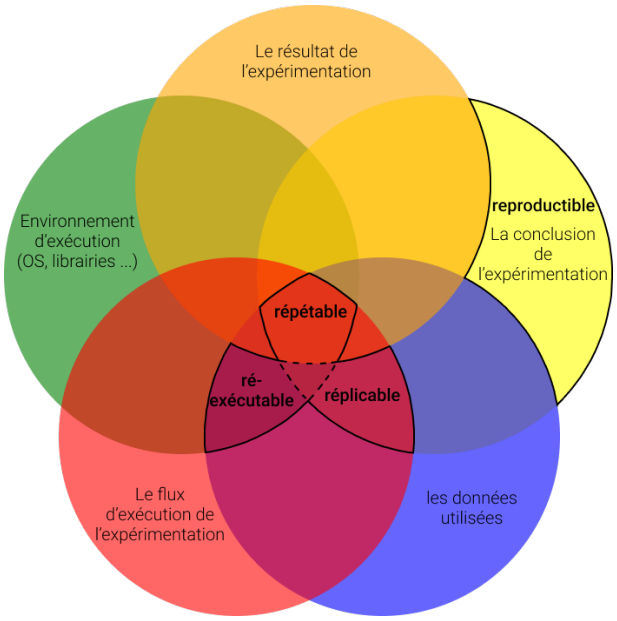

FIGURE 1 – Diagramme de Venn - Définition basée sur les 5 critères

3.2 Quelques réflexions méthodologiques

Comme toujours, bien que les définitions cherchent une caractérisation précise et exhaustive du domaine, il convient de ne pas négliger une approche pragmatique qui se focaliserait sur les pratiques acceptables par les équipes de recherche, tendant vers le compromis entre qualité des résultats, qualité des publications, coût des expérimentations, valorisation des efforts (qui parle de ROI ?). Dans cet objectif, nous avons recensé quelques pratiques, classées selon leur degré de recommandation dans ce premier travail bibliographique. Il s'agit d'une liste non-exhaustive de bonnes pratiques dont la validité n'aura de sens que par l'adhésion d'une communauté de chercheurs qui aspirent à plus de reproductibilité.

1. **Documenter le mieux possible.** Cela peut paraître trivial mais il est essentiel de documenter l'environnement original utilisé pour exécuter le code, cela inclut toutes les dépendances avec leurs versions et sous-versions. Pour faciliter la réutilisation, documenter adéquatement les artefacts computationnels. Documenter aussi les résultats, y compris ceux des expériences ratées. Les tracés quant à eux doivent être réalisés uniquement via code. [18, 24, 7, 5, 8]
2. **Automatiser au maximum, minimiser le nombre d'étapes manuelles.** Favoriser une approche DevOps afin de rendre transparentes toutes les manipulations de données. Les actions manuelles doivent être remplacées par des scripts ou enregistrées dans des fichiers. Les résultats doivent être reproduits à partir des données numériques les plus anciennes de l'expérience, qu'il s'agisse de données brutes provenant d'instruments ou d'observations.[18, 7, 22, 10, 5]
3. **Rendre accessible les codes, logiciels et datas.** Pour faciliter la reproductibilité, partager les données, les logiciels, les flux de travail et les détails de l'environnement de calcul dans des dépôts ouverts et fiables est primordial. Conserver des exemples reproductibles sur des plateformes indépendantes des institutions et partager les projets de manière redondante sur plusieurs plateformes est un bon moyen de garantir la pérennité des données. Fournir un accès public aux scripts et aux résultats des exécutions. S'assurer que le code et les données soient sous licence ouverte pour la réutilisation et distribuer la licence avec le projet (licence *Creative Commons* par exemple). Négocier les licences ouvertes pour les données et le code avec les collaborateurs avant le début du projet de recherche. Selon la taille des données plusieurs méthodologies peuvent être appliquées :
   — Petite donnée statique <2GB : Mettre les ensembles de données à disposition dans un dépôt externe dédié à l'accès aux ensembles de données scientifiques à perpétuité.
   — Grande donnée statique >2GB : Référencer et citer la version des données utilisées et fournir le code informatique pour manipuler les données.
   — Streaming Data : Les résultats publiés doivent être obtenus à partir d'une certaine quantité de données fixes.
   [18, 24, 8, 22, 21]
4. **Appliquer un contrôle de versions des artefacts externes.** Il est important d'archiver les versions exactes de tous les programmes externes utilisés et de travailler avec des services permettant la prise de « snapshots et d'images » des bases de données et des logiciels utilisés. Enfin, il faut appliquer un contrôle de version pour les environnements afin

de rendre les informations des environnements disponibles et documentées. [18, 24, 10, 22]

5. **Appliquer une politique de stockage des données de l'expérience.** Il est conseillé d'enregistrer tous les résultats intermédiaires, si possible dans des formats standardisés, ainsi que toujours stocker les données brutes sous-jacentes aux graphiques. Il est évidemment recommandé que les auteurs incluent à leurs publications, les données, les algorithmes ou toute autre information qui est centrale ou intégrale à la publication.[18, 10, 8]
6. **Identifier de façon pérenne.** Pour faciliter le référencement, les articles devraient contenir des liens permanents vers le code et/ou les données. Il est aussi intéressant d'attacher aux code et données des identifiants uniques pour assurer l'identification de la version et la traçabilité. Enfin permettre la publication des articles avec leur codes et données en un seul paquet permettrais une plus grande accessibilité aux expériences et améliorerait la reproductibilité.[22, 8, 5]
7. **Appliquer un contrôle de versions des artefacts internes.** Utiliser un contrôle de version pour tous les scripts personnalisés afin de les rendre disponibles à l'extérieur.[18, 22]
8. **Documenter les paramètres d'entrées.** Pour les analyses qui incluent de l'aléatoire, noter les graines de génération utilisées. Les valeurs d'entrée doivent être incluses avec le code et les scripts qui ont généré les résultats.[18, 22]
9. **Modulariser et découpler.** Pour améliorer la reproductibilité, il est recommandé de maximiser la modularité du projet et de minimiser les dépendances entre les composants développés. Cela permet une meilleure compréhension du code et favorise les modifications futures. [24, 10]
10. **Sourcer.** En plus des bonnes pratiques de contrôle de versions, la citation des sources et auteurs de ces outils est également recommandé. Il serait également idéal de proposer des standards de citation pour cela. [22, 8]
11. **Contrôler la version des données.** Le contrôle de version du code et des outils utilisés ou développés n'est pas suffisant, il faut également assurer un contrôle de version des données. Celles-ci peuvent grandement varier au cours du temps et il est important de garder une trace de ces changements. Lorsque l'expérience utilise des données extérieures, il est également nécessaire de préciser dans une documentation la version précise utilisée de ces données.[22]
12. **Pré-calculer si nécessaire.** Lors du cas particulier d'une expérience de simulation chronophage, il peut être intéressant de fournir un maximum de données précalculées avec le code pour faciliter la reproduction de l'expérience. [22]

3.3 Les méthodes de développement logiciels en support

Nous présentons dans cette partie les méthodes et outils de production logicielle qui peuvent soutenir une activité de recherche reproductible.

Dans l'industrie du logiciel, les logiciels sont développés, puis vérifiés, validés et qualifiés dans différents environnements de pré-production. Il s'agit bien d'une démarche de reproduction puisque l'on souhaite obtenir le même résultat dans des environnements différents [1].

Nous allons donc nous intéresser au concept de gestion de version et de configuration, qui a pour objectif d'expliciter, simplifier et automatiser le partage de code, leurs différentes versions et dépendances. Nous présentons également le concept d'intégration et de déploiement automatique, qui vise à limiter à chaque mise en production d'un nouveau programme/version l'intervention humaine.

Les outils de gestion de version, de configuration et de dépendances.
Les outils de contrôle de versions sont très connus dans le milieu du développement, et certains comme Git sont devenus un standard chez les développeurs professionnels. Ces outils permettent une gestion décentralisée du code et la définition d'un serveur de référence souvent assimilé à un dépôt centralisé. Celui-ci permet à toutes les parties prenantes de collaborer tout en assurant la synchronisation des modifications. On peut ainsi suivre chaque ajout et suppression et comparer plusieurs versions entre elles.

Un outil comme Git a également l'ambition de faire de la gestion de configurations. Le dévelop-

pement peut être organisé en plusieurs "branches", chacune dédiée à une fonctionnalité/évolution qui lui est propre (une branche bugfix pour corriger les bugs, une branche interface pour le développement de l'interface ...) et de les combiner entre elles. Pour compléter cette gestion des branches, la définition d'un workflow[25] est nécessaire pour convenir des enchaînements des activités de gestion de version et de configuration. Appliquer un workflow adapté à un projet et mettre en place un outil de contrôle de version permettent donc un meilleur suivi et une meilleur traçabilité du projet. A ce titre, ils paraissent tout à fait pertinents à appliquer dans le contexte de la reproductibilité de la recherche.

La gestion de dépendances s'intéresse aux dépendances entre les constituants d'un programme. Ces dépendances peuvent être logicielles ou environnementales. La gestion des dépendances s'intéresse à :

— quel sont les bibliothèques tierces utilisées et sous quelles versions ?
— si le programme est écrit dans un langage compilé, quel compilateur, sous quelle version et quelles options ?

Pour les bibliothèques tierces, il existe de nombreux logiciels selon le langage (ex : Maven pour Java) qui permettent de lister toutes les dépendances et de les télécharger automatiquement pour les inclure. Pour les applications, il est possible d'utiliser des outils comme Make qui permettent de lister toutes les étapes et options nécessaires à la compilation, d'automatiser le processus et de vérifier que tout s'est bien déroulé. La gestion de ces dépendances est primordiale mais est spécifique à chaque projet en fonction du langage utilisé. Il est donc primordial pour le développeur comme pour le chercheur de bien l'intégrer au projet et de la documenter

La gestion des dépendances environnementales s'intéresse à :

— quel est le système d'exploitation utilisé ?
— quels sont les logiciels et outils installés sur la machine ?

Face à l'hétérogénéité de ces conditions environnementales, les diverses solutions recherchent un niveau d'homogénéisation. Une première solution est l'utilisation d'une machine virtuelle, qui émule le fonctionnement complet d'une machine et de son système d'exploitation, isolé du système hôte sur lequel il tourne. Une seconde solution est la conteneurisation, apparue plus récemment, qui offre l'avantage d'être plus légère et rapide comparée à la virtualisation[20], puisque elle se contente de créer un environnement isolé du reste du système, mais qui tourne sur le même noyau que la machine hôte. De plus, la virtualisation permet de décrire en détail l'environnement qui sera reconstruit à partir de celui-ci, ce qui permet de le rendre à la fois moins opaque et plus léger, ce qui en fait donc une technologie idéale à intégrer dans des recherches reproductibles.[7]

Intégration et déploiement automatique. Dans la culture DevOps[17] (terme issu de la contraction des termes "Développement" et "Opérationnalisation"), l'intégration et le déploiement automatique ont pour objectif de regrouper automatiquement des parties développées séparément, de les déployer automatiquement sous la forme d'un nouvel artefact si celui-ci est conforme aux exigences.

Cette chaîne de production se compose généralement de divers outils qui intègrent, vérifient, valident et qualifient les différents constituants d'un produit logiciel. Dans l'idéal, une telle chaîne, quand elle est automatisée, amène un haut niveau de qualité et de garanties de la production [26]. Dans un contexte de développement logiciel commercial, l'objectif est de réduire le "time-to-market" en réduisant les délais de production tout en minimisant les erreurs. Dans un contexte de recherche d'une meilleure reproductibilité, l'explicitation des différentes étapes, dans un formalisme informatique et donc non ambigu, apporte transparence et traçabilité à la démarche empirique de production. Grâce à la disponibilité de ce type d'artefacts, l'expérience est également mieux réutilisable et modifiable.

En combinant les outils présentés ci-dessus, il est alors possible d'adresser plusieurs éléments techniques favorisant la reproductibilité, comme nous le verrons dans la partie suivante. Par exemple, la recherche systématique des versions les plus récentes des constituants environnementaux est contradictoire avec la définition de répétabilité et de réplicabilité car il s'agit au contraire de "bloquer" les versions utilisées. Dans un contexte de recherche en intelligence artificielle, le corpus de données servant à la phase d'exploration ne doit pas entraver les besoins de répétabilité et de réplicabilité.

4 Une première approche vers la reproductibilité

En se basant sur les définitions et outils vus plus haut, nous avons proposé dans [**carayon_jfms22**] quelques réflexions méthodologiques à travers la définition de processus et d'outils à appliquer pour tendre vers une démarche reproductible. Cette démarche est rappelée ici puis elle est illustrée par une petite expérience montrant comment accéder à différents niveaux de reproductibilité.

4.1 La démarche

Fournir un workflow adapté à la recherche. Pour prendre en compte la démarche itérative propre à l'approche scientifique et assurer la traçabilité entre développement du prototype et expérimentation, un processus de controle de versions et de configurations est défini.

Identifier et expliciter les paramétrages de l'application. Il s'agit de capturer les manipulations de l'expérimentateur afin d'obtenir le résultat de l'expérience. Celles-ci peuvent être stockées dans des fichiers de paramétrage qui sont ensuite identifiés et renseignés. Enfin, il est important que tous les appels à des calculs stochastiques soient répertoriés et tracés. Il s'agit alors d'enregistrer les racines initialisant ces calculs car elles constituent elles aussi une variable impactant le résultat de l'expérience.

Identifier et expliciter les données utilisées et produites. Les données utilisées et produites sont une partie névralgique d'une expérimentation. Elles doivent faire l'objet d'une attention particulière en fonction de leur localisation, de leur acheminement, de leur taille. Dans tous les cas, les données doivent être traçables et versionnées pour assurer la répétabilité.

Automatiser ces étapes à l'aide d'un outil. Cela nous a amené à proposer un prototype qui permet d'automatiser une partie des ces activitées et qui, par sa forme, pousse un chercheur à adapter des bonnes pratique en terme de gestion des données. Dans un premier temps, l'outil va d'abord vérifier automatiquement la forme d'un projet en terme de workflow pour que celui-ci convienne aux bonnes pratiques d'une expérimentation. Le programme s'occupe ensuite de demander à l'expérimentateur

- où sont les données requises
- où sont les données produites
- où sont les fichiers de paramètres de l'expérimentation
- de répliquer l'expérience dans le terminal

Le programme peut ensuite automatiquement stocker toutes les informations nécessaires pour répliquer l'expérience et pousser les résultats dans une branche dérivée du code source. Pour reproduire l'expérience, il suffit de fournir au programme l'URL de l'expérimentation correspondante et le programme va automatiquement, à partir des informations enregistrées, la reproduire et vérifier que le résultat est bien identique à l'expérimentation originale. Celui-ci va aussi vérifier que le point de départ de l'expérimentation est correctement étiqueté et que les seules modifications qui ont été effectuées sont sur les entrées, sorties et paramètres. Enfin, le programme va pousser l'expérimentation et ses résultats dans une nouvelle branche dérivée de la branche initiale, avec une étiquette qui permet de facilement l'identifier et de remonter la source.

4.2 Présentation d'une expérimentation

Pour mettre en évidence les problèmes rencontrés pour la reproductibilité, nous avons mis en place une expérience simple qui montre les conséquences sur un projet simple.

Niveau 0. Il s'agit du niveau le plus bas dans lequel le code source est présent mais formellement rien n'est mis en place pour la reproductibilité. [4] Malgré le fait que le code ne fasse que

```python
import matplotlib.pyplot as plt
import numpy as np
import random
import sys

def experiment(tries,min=0,max=10) -> None:
    numbers = []
    for _ in range (tries) :
        numbers.append(random.randint(min,max))
    print("Generated :" + str(numbers))

    plt.hist(numbers,range=(min,max),bins=max+1,color='blue',edgecolor='black')
    print(f"average : {np.mean(numbers)}")
    plt.show()

if __name__ == '__main__':
    if len(sys.argv) == 2 :
        experiment(int(sys.argv[1]))
    elif len(sys.argv) == 4 :
        experiment(int(sys.argv[1]),int(sys.argv[2]),int(sys.argv[3]))
```

FIGURE 2 – Code source niveau 0

20 lignes, on compte dans cet exemple beaucoup de freins à la reproductibilité.

4. https://gitlab.irit.fr/smac/reproductibility/reproductibiliteniveau0

— des librairies externes sont utilisées et doivent donc être installées avec la bonne version
— la fonction utilise un "type hint", présent depuis python 3.5, dont le code ne fonctionnera pas sur une version antérieure
— l'utilisation d'un tirage aléatoire va produire des résultats différents à chaque ré-exécution
— les entrées générées et le résultat sont affichés dans la console, ce qui nécessite de noter manuellement chaque résultat
— le fonctionnement et le paramétrage de l'expérimentation n'est décrit nulle part, l'expérimentateur doit le trouver lui même
— aucun workflow n'est présent, on ne sait pas quelle version on utilise, ni quelles sont le ou les expérimentations

Niveau 1. Ici, on se place dans un contexte de ré-exécutabilité. On cherche à mieux décrire le processus de l'expérimentation et rendre consistant des exécutions successives. [5]

— on a séparé la génération des inputs et des outputs et un fichier "main" s'occupe de l'exécution, mettant en évidence le flux d'exécution de l'application
— une branche pour le code source est présente ainsi que deux autres branches qui décrivent deux expérimentations différentes

Cela permet de récupérer une expérimentation précise et de la relancer. C'est encore insuffisant car le workflow reste faible : si l'on rajoute d'autre évolutions, elles risquent de cacher le travail actuel, et les résultats entre deux exécutions restent différents.

Niveau 2. L'objectif est, ici, d'obtenir une meilleure réplicabilité de l'expérimentation. On va donc mettre en évidence les paramétrages de l'application, ainsi que les données en entrée et en sortie. [6]

— on utilise un fichier yaml pour les paramètres qui sont décrits de façon explicite ; on ne modifie plus le code pour chaque nouvelle exécution.
— on prend maintenant en compte la gestion de l'aléatoire avec la mémorisation de la graine qui permet d'obtenir des résultats consistants
— les données générées et la sortie sont maintenant stockées et ne sont plus juste affichées dans la console, ce qui facilite les comparaisons entre plusieurs expériences
— le processus est amélioré avec un système d'étiquettes. Chaque version fixe du programme est étiquetée, chaque expérimentation possède elle aussi une étiquette, dérivée de celle du programme.

On obtient donc des exécutions qui sont consistantes entre plusieurs essais. Les données en entrées, sorties et les paramètres sont explicités et accessibles. Le processus décrit chaque expérimentation.

Niveau 3. A ce niveau, on s'intéresse à la répétabilité, on souhaite donc que quelqu'un puisse répéter une expérience dans des conditions identiques à celles de départ. [7]

— on rajoute un conteneur (ici en Docker) avec des versions fixées (par opposition à la pratique usuelle qui consiste à récupérer la dernière version). De cette façon, on est certain que les dépendances sont respectées et fonctionnent
— on ajoute à chaque expérimentation les entrées et sorties originelles
— pour chaque expérimentation, on rajoute des tests qui vérifient que ce que ce qui est produit avec la future ré-exécution est identique à celle originelle.

Ici, l'expérimentateur futur bénéficie d'un processus clairement défini. Les différentes versions du programme et de l'expérimentation sont clairement identifiées. Les données en entrée, sortie et les paramètres sont eux aussi identifiés. Un fichier Docker permet de s'abstraire du besoin de vérifier manuellement la compatibilité avec notre environnement car les logiciels sont toujours installés avec la même version. Enfin, on peut comparer facilement le résultat d'une ré-exécution avec celle originelle pour voir si tout s'est correctement déroulé.

5 Discussion et perspectives

Nous avons que le dernier niveau semble offrir une répétabilité complète qui permet donc à l'utilisateur futur de reproduire à l'identique une expérimentation. Cependant, il reste encore beaucoup de manipulations de l'expérimentateur

5. https://gitlab.irit.fr/smac/reproductibility/reproductibiliteniveau1
6. https://gitlab.irit.fr/smac/reproductibility/reproductibiliteniveau2
7. https://gitlab.irit.fr/smac/reproductibility/reproductibiliteniveau3

original qui lui permettent de mettre en place la démarche reproductible. En effet ce n'est que par la rigeur d'une démarche sans faille que l'expérimentateur obtient :

— un workflow permettant la traçabilité et la reproduction des étapes
— l'identification et l'historisation des données en entrées/sorties et des paramètres
— la consistance du code entre développement et expérience
— la mise à disposition des entrées et sorties originales
— la mise à disposition des tests pour vérifier que les résultats d'un ré-exécution sont bien identiques

A l'inverse, l'automatisation par un outil d'assistance [8] à la démarche reproductible tel que nous l'avons décrit en 4.1 permet d'obtenir, dans notre proposition, un niveau comparable au niveau 3 et de bénéficier de trois avantages.

— l'automatisation ou la recommandation des étapes d'un processus de reproductibilité
— la conformité à une démarche reproductible
— la simplification des activités de répétabilité ou de réplicabilité

Ceci n'est évidemment qu'une première étape. Il faut travailler selon les cinq axes principaux dont nous avons souligné l'importance pour la reproductibilité : l'environnement, les données en entrées, les données en sortie, le flux d'exécution et la conclusion scientifique. Ce travail met en évidence ces cinq éléments mais il souligne également la réflexion à produire sur les outils tout autant que sur les démarches à mettre en oeuvre.

La prochaine étape consiste donc à appliquer ces artefacts sur des travaux de thèse en cours afin d'en étudier l'applicabilité et l'adaptabilité face à la variété des travaux. Les travaux sur les systèmes multi-agents, leurs connexions aux sciences du complexe, vont certainement lever des questions de répétabilité et de réplicabilité dans un domaine caractérisé par le déterminisme non prédictif des systèmes complexes. Cela nous pousse à questionner notre démarche de chercheur en système multi-agent, empirique par nature, et à la justifier par rapport à nos objectifs d'acteur de la recherche publique. Passionnant.

8. https://github.com/AxelCarayon/reproductionTool

6 Remerciements

Nous remercions les étudiants du master Informatique de l'Université Toulouse 3 Paul Sabatier Avi Amzalac, Gautier Perez, Gregory Caderby, Tom Meehan pour leur travail préparatoire. Nous remercions également le parcours Science Du Logiciel du master Informatique pour le financement de ce travail dont l'objectif est également l'amélioration de la formation pour la reproductibilité.

7 Bibliographie

(1) Anda, B. C. et al. (2009). Variability and Reproducibility in Software Engineering : A Study of Four Companies that Developed the Same System. *IEEE Transactions on Software Engineering 35*, Conference Name : IEEE Transactions on Software Engineering, 407-429.

(2) Arcangeli, J.-P. et al. in *Architectures logicielles : principes, techniques et outils - Traité RTA, série Informatique et Systèmes d'Information* ; Série Informatique et Systèmes d'Information, ISSN 2111-0360 Chapter 10, t. 10 ; Hermès Science - Lavoisier : 2014, p. 1-30.

(3) Baker, M. (2016). Reproducibility crisis. *Nature 533*, 353-66.

(4) Barba, L. A. (2018). Terminologies for Reproducible Research. *arXiv :1802.03311 [cs]*, arXiv : 1802.03311.

(5) Barba, L. A. (2016). The hard road to reproducibility. *Science 354*, Publisher : American Association for the Advancement of Science Section : Working Life, 142-142.

(6) Benureau, F. C. Y. et al. (2018). Re-run, Repeat, Reproduce, Reuse, Replicate : Transforming Code into Scientific Contributions. *Frontiers in Neuroinformatics 11*.

(7) Boettiger, C. (2015). An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Operating Systems Review 49*, arXiv : 1410.0846, 71-79.

(8) Brinckman, A. et al. (2019). Computing environments for reproducibility : Capturing the "Whole Tale". *Future Generation Computer Systems 94*, 854-867.

(9) Cohen-Boulakia, S. et al. (2017). Scientific workflows for computational reproducibility in the life sciences : Status, challenges and opportunities. *Future Generation Computer Systems 75*, 284-298.

(10) Cole, B. S. et al. (2018). Eleven quick tips for architecting biomedical informatics workflows with cloud computing. *PLOS Computational Biology 14*, Publisher : Public Library of Science, e1005994.

(11) Ferber, J., *Multi-agent systems - an introduction to distributed artificial intelligence* ; Addison-Wesley-Longman : 1999.

(12) Goodman, S. N. et al. (2016). What does research reproducibility mean ? *Science Translational Medicine*, Publisher : American Association for the Advancement of Science, DOI : 10.1126/scitranslmed.aaf5027.

(13) Gundersen, O. E. et al. (2018). State of the Art : Reproducibility in Artificial Intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence 32*, Number : 1.

(14) Hunold, S. et al. (2013). On the State and Importance of Reproducible Experimental Research in Parallel Computing. *arXiv :1308.3648 [cs]*, arXiv : 1308.3648.

(15) Hutson, M. (2018). Artificial intelligence faces reproducibility crisis. *Science 359*, Publisher : American Association for the Advancement of Science, 725-726.

(16) Ince, D. C. et al. (2012). The case for open computer programs. *Nature 482*, 485-488.

(17) Jabbari, R. et al. in *Proceedings of the Scientific Workshop Proceedings of XP2016*, Association for Computing Machinery : New York, NY, USA, 2016, p. 1-11.

(18) Sandve, G. K. et al. (2013). Ten Simple Rules for Reproducible Computational Research. *PLOS Computational Biology 9*, Publisher : Public Library of Science, e1003285.

(19) Sève, L. et al., *Émergence, complexité et dialectique : sur les systèmes dynamiques non linéaires* ; Odile Jacob : 2005.

(20) Sharma, P. et al. in *Proceedings of the 17th International Middleware Conference*, ACM : Trento Italy, 2016, p. 1-13.

(21) Stodden, V. (2009). The Legal Framework for Reproducible Scientific Research : Licensing and Copyright. *Computing in Science Engineering 11*, Conference Name : Computing in Science Engineering, 35-40.

(22) Stodden, V. et al. (2014). Best Practices for Computational Science : Software Infrastructure and Environments for Reproducible and Extensible Research. *Journal of Open Research Software 2*, Number : 1 Publisher : Ubiquity Press, e21.

(23) Stodden, V. et al. (2013). Toward Reproducible Computational Research : An Empirical Analysis of Data and Code Policy Adoption by Journals. *PLoS ONE 8*, sous la dir. de Zaykin, D., e67111.

(24) Tatman, R. et al. (2018). A Practical Taxonomy of Reproducibility for Machine Learning Research.

(25) van der Aalst, W. et al. (2003). Workflow Patterns. *Distributed and Parallel Databases 14*, 5-51.

(26) Zhao, Y. et al. The impact of continuous integration on other software development practices : A large-scale empirical study, en-US, 2017.