Comprendre et générer des distributions de commandes réalistes

Julien Gori, Mohamed Ali Ben Amara, Gilles Bailly

Comprendre et générer des distributions de commandes réalistes
Understanding and generating realistic command distributions

Julien Gori
julien.gori@sorbonne-universite.fr
Sorbonne Université, CNRS, Institut des Systèmes Intelligents et de Robotique, ISIR
Paris, France

Mohamed Ali Ben Amara
mohamed-ali.ben-amara@ensta-paris.fr
ENSTA Paris, IPP
Palaiseau, France

Gilles Bailly
gilles.bailly@sorbonne-universite.fr
Sorbonne Université, CNRS, Institut des Systèmes Intelligents et de Robotique, ISIR
Paris, France

RÉSUMÉ

Dans le monde réel, les séquences de commandes exécutées par un utilisateur sont complexes, alors même que les études (empiriques, simulations de systèmes intelligents) utilisées en IHM sont fortement simplifiées : par exemple des séquences indépendantes distribuées selon une loi uniforme ou de Zipf. Dans cet article, on décrit d'abord les caractéristiques des distributions de commandes réelles, en étudiant les dépendances entre commandes à travers des modèles de Markov. On propose ensuite un algorithme pour générer des séquences de Zipf avec des dépendances dont les performances sont évaluées puis discutées.

ABSTRACT

In the real world, the sequences of commands encountered by a user are complex, whereas studies in HCI (empirical, intelligent systems simulations) dealing with commands generally use simplified stimuli: for example, independent sequences distributed according to a uniform or Zipf distribution. In this article, we first describe the characteristics of real commands, studying the dependencies between commands using Markov models. We then propose an algorithm for generating Zipf sequences with dependencies, which we evaluate and discuss.

CCS CONCEPTS

• **Human-centered computing → HCI theory, concepts and models**.

MOTS CLÉS

Loi de Zipf, séquence de commandes, évaluation

KEYWORDS

Zipf's law, command sequence, evaluation

Reference:
Julien Gori, Mohamed Ali Ben Amara, and Gilles Bailly. 2024. Comprendre et générer des distributions de commandes réalistes.

IHM '24, March 25–29, 2024, Paris, France
2024.

1 INTRODUCTION

Les études empiriques ou de simulation faisant intervenir des séquences de commandes [1] sont courantes en IHM ; on recense par exemple de nombreuses expériences contrôlées pour évaluer des techniques de menus [4, 8, 9, 11, 23, 27] des systèmes intelligents, p.ex. [30] ou de recommendation [25]. Au cours d'une expérience contrôlée typique, on demande aux participants de sélectionner des séquences de commandes, les une à la suite des autres. On y retrouve essentiellement deux types de séquences. Les premières sont les séquences uniformes : toutes les commandes sont présentées le même nombre de fois au participant de l'expérience. Les deuxièmes sont celles qui s'appuient (de près ou de loin) sur la loi de Zipf — des séquences qu'on appellera *zipfiennes*. La loi de Zipf décrit une loi puissance qui lie la fréquence d'un objet (ici une commande) et son rang : Il y a peu de commandes très fréquentes, et la plupart des commandes sont rarement utilisées. Comme on le verra sous-section 2.1, les séquences zipfiennes sont plebiscitées par souci de validité écologique, vu que les séquences réellement observées se conforment à la loi de Zipf.

Ce choix de séquence pose au moins deux problèmes : Tout d'abord il n'est pas toujours clair quel type de séquence un chercheur doit choisir, entre une séquence uniforme ou une (pseudo)zipfienne, et quel sera l'impact de ce choix sur la simulation d'un système ou son évaluation empirique. Ensuite, certaines commandes ont de fortes dépendances. Par exemple *copier* sera presque tout le temps suivie de *coller*. Or les séquences zipfiennes utilisées habituellement sont indépendantes *i.e.*, l'ensemble des positions d'une même commande dans la séquence ne dépend que de sa fréquence telle qu'indiquée par la loi de Zipf, mais pas des commandes qui précèdent. Les séquences zipfiennes sont donc perfectibles du point de vue de la validité écologique.

Cet article adresse ces deux problèmes. Tout d'abord, nous recensons divers arguments pour guider un expérimentateur vers l'utilisation d'un certain type de séquence. Ces arguments sont utiles au moment de la conception d'expérience contrôlée, et peuvent servir de justificatifs face à des relecteurs critiques. Ensuite, nous modélisons les dépendances entre commandes avec un modèle de Markov simple. Nous montrons notamment que les bigrammes de commandes présentés sur une séquence réaliste sont différents

1. Par commande, on entend toute action dont l'utilisateur est à l'initiative ; celle ci peut-être déclenchée par un bouton, par sélection d'un élément dans un menu, d'un raccourci clavier *etc.*

des bigrammes retrouvés pour des séquences zipfiennes indépendantes, et nous proposons un algorithme générant des séquences zipfiennes avec dépendences qui résulte en des bigrammes plus proches de ceux observés empiriquements. Nous finissons par discuter les implications de ces résultats pour l'évaluation des IHMs qui font intervenir des séquences de commandes. L'ensemble des algorithmes (calcul, ajustement et visualisation des bigrammes et des matrices de transition, et algorithme pour générer les séquences) est disponible https://github.com/jgori-ouistiti/ihm24-zipf/.

2 ÉTAT DE L'ART

Nous discutons les travaux précédents étudiant les distributions de commandes dans des tâches réelles ou ceux les générant en IHM.

2.1 Les distributions de commandes dans les tâches réelles : Loi de Zipf

La loi de Zipf lie la probabilité d'apparition $f(s,k)$ d'une observation à son rang [2] k selon une loi puissance

$$f(s,k) = C\frac{1}{k^s}, \tag{1}$$

où C est une constante de normalisation. Le paramètres s indique à quel point les observations sont inégalement réparties : pour $s = 0$, la loi de Zipf se comporte comme une loi uniforme [3] ; plus s est grand, plus les observations des rangs inférieurs sont fréquents comparativement aux autres. Pour évaluer si les données suivent bien une loi de Zipf, les données sont souvent représentées dans un plan log-log : en effet $\log f(s,k) = \log C - s\log(k)$, et donc $\log f(s,k)$ en fonction de $\log(k)$ suit théoriquement une droite avec pour ordonnée à l'origine $\log C$ et pour pente $-s$.

La loi de Zipf est encore relativement mal comprise et il n'est pas tout à fait clair quelles sont les propriétés qui font qu'un ensemble d'objets sera distribué selon une loi de Zipf [4]. En particulier, si des ensembles d'objets suivent une loi de Zipf, leur union ou des sous ensembles n'en suivent souvent pas une. Par exemple, si les tailles des principales villes en France, en Italie, en Allemagne *etc.* suivent approximativement une loi de Zipf, ce n'est pas du tout le cas pour les principales villes à l'échelle de l'Union Européenne [5].

Ceci dit, que les commandes soient souvent distributées suivant une loi de Zipf est connu depuis longtemps. C'est ce que montrent plusieurs travaux dès les années 80, essentiellement basés sur des commandes Unix éxécutées dans un *shell*, mais aussi sur un éditeur de texte [18]. Par exemple, plusieurs travaux référencés dans [12, Section 4] démontrent qu'une loi de Zipf rend fidèlement compte des commandes utilisées par une population entière. Toujours sur Unix, il est aussi montré que les fréquences des même commandes peuvent être différentes parmi des groupes d'utilisateurs différents, quand bien même ces fréquences s'accordent avec la loi de Zipf [12].

2. Le rang d'une commande est sa position quand les commandes sont classées par ordre décroissant de fréquence d'apparition : la commande la plus fréquente à le rang 1, la deuxième commande la plus fréquente à le rang 2 et ainsi de suite.

3. Cette propriété de la loi de Zipf semble contradictoire avec l'introduction, où on fait la différence entre la loi de Zipf et celle uniforme. En pratique les commandes suivent une loi de Zipf avec $s > 1$, et on est très loin du comportement uniforme ; séparer les deux cas de figure semble plus clair.

4. L'article Wikipedia sur la loi de Zipf donne plusieurs explications potentielles, sans qu'il y en ait une de conclusive.

À partir d'une analyse des commandes éxecutées dans une application d'édition d'image par manipulation directe, Lafrenière *et al.* [20] montrent également un faible taux de recouvrement de commandes entre utilisateurs : Seulement 15 commandes sont communes à au moins 50% des utilisateurs, et 257 commandes sont communes à moins de 10% des utilisateurs (sur 352 commandes). Des résultats similaires sur population entière sont aussi trouvés sur *MS Word* [8, 22]. Ainsi, de nombreux exemples venant de la ligne de commande ou de logiciels de manipulation directe mettent en évidence que le lien entre la fréquence d'utilisation des commandes et leur rang est bien modélisable par une loi de Zipf.

2.2 Les distributions de commandes dans les tâches réelles : Dépendances entre commandes

Les dépendances entre commandes, bien que reconnues [29], sont moins bien documentées. Hanson *et al.* [14, Figure 4] montrent un graphe des commandes Unix et leur dépendances, par exemple des commandes d'information (*e.g.*, *ls* qui liste les fichiers dans le dossier courant) suivent souvent des commandes de positionnement (*e.g.*, *cd* qui change le dossier courant), avec des dépendances qui peuvent s'étendre sur plus de quatre commandes. On peut aussi citer encore une fois les travaux de Greenberg [12] qui note une dépendance temporelle : pour les commandes Unix, il y a une très forte probabilité que la commande qui vient soit une des sept dernières à avoir été éxecutées.

En dehors de la communauté IHM, on trouve des modèles de Markov pour représenter les dépendances entre commandes. Par exemple, un modèle de Markov est utilisé pour modéliser les dépendances entre commandes propres à un utilisateur, et ce à des fins de détection d'intrusion, avec un modèle d'ordre 1 [31] voir d'ordre 10 [16] *i.e.*, les dix dernières commandes sont supposées être informatives pour prédire l'apparition de la prochaine, bien que les auteurs ne justifient pas ce chiffre. Enfin dans le cadre de la prédiction de commandes (*e.g.*, par des systèmes de recommendation) les dépendances entre commandes ont aussi été modélisées. On retrouve aussi des modèles de Markov d'ordre 1 [1], mais aussi des modèles plus complexes qui aggrègent plusieurs modèles de Markov (voir [15] pour plusieurs exemples), ou des réseaux de neurones profonds [33].

2.3 Les séquences de commandes utilisées en IHM

En IHM, nous avons trouvé trois modes de création de séquences de commandes utilisées lors d'expériences contrôlées, a savoir les séquences uniforme et zipfienne précédemment citées, et pour lesquelles on renvoie vers [23] pour de nombreuses références, et les séquences basées sur des vrais séquences de commandes [8, 11]. Par exemple Findlater *et al.* [8] se basent sur des séquences acquises sur *MS Word* d'utilisateurs en condition de travail normales, et remplacent les commandes originales par des commandes abstraites (villes, boissons *etc.*). Souvent, les distributions réellement utilisées peuvent être quelque peu différentes des distributions théoriques. Par exemple, certains expérimentateurs ne demandent la sélection que de certaines commandes [4, 23], et les fréquences exactes sont parfois altérées ; par exemple dans [23], les fréquences

de commande pour la séquence uniforme sont inégales tout en restant proches. Une autre variation autour de la loi de Zipf qui a été répliquée plusieurs fois est la "double zipfienne" de Grossman *et al.* [13], qui met en parallèle deux zipfiennes (les deux moitiés des commandes sont distribuées selon la même loi zipfienne.)

2.4 Le cas de l'entrée de texte : pourquoi une différence ?

De l'avis de Greenberg et Witten [12], dans les années 90 les travaux qui se focalisent sur l'entrée de texte sont plus avancées dans la modélisation des séquences (ici de caractères) que ceux qui se penchent sur les séquences de commandes. Greenberg et Witten citent notamment l'exemple d'un clavier adaptatif qui prédit des caractères [6]. Le cas de l'entrée de texte est intéressant à mentionner ici ; en effet bien qu'il peut sembler quelque peu éloigné au premier abord, les problèmes rencontrés par l'entrée de textes ont plusieurs similarités avec la sélection de commandes. La distribution des mots suit aussi une loi de Zipf — en fait George Zipf, dont la loi porte le nom, était un linguiste [10] [5] — avant de trouver des applications dans d'innombrables autres domaines. Et tout comme les commandes, il existe des fortes dépendances entre lettres et mots qui se suivent, comme illustré par exemple par Shannon [28, Section 3]. Il semble naturel pour la plupart des chercheurs d'évaluer une technique d'entrée de texte avec un stimulus qui respecte les dépendances de la langue, et d'ailleurs nous ne connaissons aucune expérience qui évaluerait une technique d'entrée de texte avec une séquence uniforme de caractères.

La question est de savoir pourquoi ce n'est pas le cas actuellement dans les évaluations de systèmes sur la base de séquences de commandes. Nous voyons plusieurs raisons à celà. D'abord, il est facile de créer un stimulus avec les bonnes propriétés de dépendances dans le cadre du texte — il suffit de choisir une phrase correcte du point de vue linguistique. On peut aussi choisir la phrase en question pour éviter des répétitions de certains caractères [24], sans vraiment plus de difficultés. Donc le stimulus est facilement contrôlable. Au contraire, actuellement, il n'existe pas à notre connaissance de méthode ou protocole en IHM facilement contrôlable pour créer des séquences de commandes qui soient réalistes, mis à part utiliser une séquence de commandes réellement entrée par un utilisateur. La deuxième raison vient de la combinaison de deux observations :

(1) Afin que la connaissance préalable des commandes puissent influencer l'évaluation, on utilise souvent des commandes abstraites [8, 27] (noms de villes, d'animaux *etc.*). Celà implique que les dépendances entre commandes, pour qu'elles aient un effet sur l'utilisateur, doivent être apprises par lui/elle. Par exemple, des commandes avec fortes dépendances, comme copier et coller, vont être souvent apprises comme une paire de commande, et exécutées en tant que tel.

(2) la séléction de commandes est très lente comparée à l'entrée de texte, en conséquence, les dépendances entre commandes seront forcément moins *visibles*, et on peut légitimement se demander quel sera l'effet mesurable sur les participants de l'expérience. Par exemple, on peut trouver des techniques d'entrée de texte où un utilisateur lambda atteint facilement 50 mots par minutes, ce qui fait approximativement 300 caractères par minutes — En prenant une taille de mot de 4,5 caractères [24], et en comptant les espaces ; le calcul est bien sur imprécis mais donne un ordre de grandeur — alors qu'on peut typiquement sélectionner plutôt autour d'une dizaine de commandes par minute (par exemple, uniquement le temps d'exécution du geste dans les marking menus lors de l'expérience de Kurtenbach et Buxton [19] est compris entre 0.75 et 5 secondes).

Donc, il n'est pas évident qu'utiliser des commandes avec des distributions à fréquences inégales ait toujours un impact mesurable lors d'une expérience contrôlée à durée limitée.

Le but de cet article est donc de faciliter l'utilisation de séquences de commandes zipfienne avec dépendances.

3 QUELS CRITÈRES POUR CHOISIR SA DISTRIBUTION DE COMMANDES ?

Uniforme ou (quasi)zipfienne : Comment choisir sa distribution de commandes dans une étude ? Nous donnons ici plusieurs arguments à considérer pour effectuer ce choix, en se basant sur notre expérience personnelle et des échanges que nous avons pu avoir avec des collègues ; en particulier, nous ne considérons pas ces critères comme exhaustifs.

Contrôle. Il existe de nombreux cas où l'on veut contrôler la fréquence associée à chaque commande. En particulier, on peut vouloir que plusieurs commandes aient la meme frequence pour étudier d'autres effets que ceux de la fréquence, comme par exemple un arrangement spatial, les caractéristique d'un geste *etc.*

- Par exemple, dans Liu *et al.* [23] des séquences quasi uniformes et zipfiennes sont construites de sorte à ce que certaines commandes aient les même fréquences, pour comparer au niveaux de ces commandes l'effet des différentes distributions.
- Il y a une interaction entre la fréquence des commandes et leur mémorisation puisque plus une commande sera répétée, plus l'utilisateur se souviendra facilement de cette dernière [26].
- Il peut y avoir une interaction entre la fréquence des commandes et leur position spatiale. Par exemple, dans un menu linéaire, on sait que les premières et dernières positions du menu sont privilégiées [2]. Une mesure de taille d'effet sous forme de temps d'execution moyenne sera donc sensible à la fréquence dans ce cas.

Un autre cas est lorsque les chercheurs répliquent un protocole pour faire une comparaison avec une étude existante.

Validité externe. La validité externe d'une expérience pose la question de la générabilité : "à quel point l'effet mésuré dans le laboratoire s'applique-t'il en dehors du laboratoire ?" [3]. Les distributions de commandes peuvent avoir plusieurs effets mesurables. Par exemple, les chercheurs peuvent comparer le temps d'éxecution moyen de deux techniques d'interaction. Or, le temps d'éxécution moyen dépend de la séquence de commandes si le temps d'éxécution de chaque commande n'est pas identique. De manière générale, une distribution plus réaliste permet donc d'avoir une meilleure

5. Zipf ne réclame pas la découverte de cette loi, qui serait en fait dûe au sténographe Jean-Baptiste Estoup, cf l'article Wikipedié sur la loi de Zipf. La loi de Zipf est donc d'autant plus liée à l'entrée de texte.

estimation de la taille d'effet en conditions réelles. Ceci est d'autant plus important lorsque les chercheurs évaluent un système qui exploite de la prédiction de commandes (p.ex. [30]). En effet, dans ce cas utiliser une séquence zipfienne indépendante "tue" toute l'amélioration potentielle apportée par le système, puisque les commandes étant indépendantes les une des autres, il n'y a rien de mieux à faire que de suggérer tout le temps la commande la plus fréquente.

Compliance avec les tests statistiques. Les séquences uniformes ont l'avantage de produire un nombre identique d'observations par commandes — les données sont alors dites équilibrées, ce qui est un prérequis pour de nombreux tests statistiques. Cet argument cependant doit être examiné avec attention. En effet, il y a un compromis à faire entre une vision utilitariste des statistiques "les outils statistiques devraient être au service de mon étude" et une vision pragmatique de la conception d'expérience : "je conçoit mon expérience pour qu'elle soit plus facile à analyser ensuite". De plus, il existe des méthodes statistiques pour compenser les données désiquilibrées. Par exemple, le t-test de Student peut (doit) être remplacé par le t-test de Welch quand le nombre d'échantillons dans les deux groups comparés sont différents [7], et certaines méthodes statistiques, comme l'ANOVA, sont réputées robustes aux différences modérées de nombre d'échantillons dans chaque groupe [32].

Puissance statistique. La puissance statistique est associée au taux d'erreur de type II *i.e.*, le taux auquel on est incapable de rejeter l'hypothèse nulle, alors même que cette dernière serait fausse. Par exemple, considérons le cas où nous cherchons à montrer la significativité d'une différence de moyenne de temps d'éxecution entre deux commandes avec un t-test. Si une des commandes apparaît très rarement (par exemple parce c'est une commande peu fréquente dans une distribution de Zipf), alors l'erreur standard associée à cette commande sera probablement importante, ce qui aura tendance à mécaniquement baisser la puissance statistique du t-test.

En résumé, le choix d'une séquence de commandes n'est pas trivial ; il dépend de l'hypothèse testée lors de l'expérience contrôlée, et peut prendre en compte la puissance statistique et l'interprétabilité des résultats : possibilité de les comparer à d'autres études ou de généraliser la taille d'effet estimée par l'expérience contrôlée. Pour mieux comprendre l'influence d'un choix de conception (par exemple, le choix des couleurs des items dans un menu), il peut être intéressant de contrôler la distribution de commandes.Par contre, utiliser une distribution réaliste permettra une ébauche de réponse à (par exemple) la question : "quel est véritablement le gain d'un menu coloré dans la vraie vie". Or actuellement, il n'est pas évident d'utiliser des séquences réalistes : il est possible de générer facilement des séquences zipfiennes *indépendantes i.e.*, où la position d'une commande dans la séquence n'a pas de lien avec celles qui précèdent, mais c'est beaucoup plus dur de générer des séquences avec dépendences similaires à la vraie vie. D'ailleurs, nous n'avons trouvé aucune expérience contrôlée conduite en IHM avec des séquences avec dépendances, mis à part celle de Findlater *et al.* [8], mais qui utilise des vraies séquences.

Utiliser des vraies séquences n'est pas forcément satisfaisant ; elles sont faiblement controlables et il est dur d'écarter à priori des séquences pathologiques. Par exemple, en entrée de texte, on est capable de reconnaître que la phrase "un quizz sur le whysky a Koweït" est peu typique — elle emploie beaucoup de consonnes rares. On est capables d'identifier cela car nous sommes sur-entraînés sur le texte et repérons facilement ces motifs, ce qui n'est pas le cas pour les séquences de commandes.

Nous introduisons maintenant un algorithme pour générer des séquences zipfiennes avec dépendances. Nous présentons d'abord les méthodes que nous utilisons pour décrire les séquences générées, qui seront comparées à celle d'un jeu de données de 9 séquences.

4 MÉTHODES

Dans cette section, nous décrivons le jeu de données utilisé, son pré-traitement, ainsi que deux méthodes de visualisation des métriques utilisées dans ce travail.

4.1 Jeu de données

Cette étude se base sur un jeu de données de commandes Unix publiquement disponible [21] que nous nommons Unix-Irvine. Les données consistent en 9 séquences de commandes, entrées dans un *shell* Unix acquises sur un ensemble de deux ans par 8 utilisateurs différents. La séquence de commandes "brute" comporte des marqueurs indiquant les débuts et fin de sessions, et des symboles génériques pour anonymiser les séquences (noms de fichiers, adresses électroniques, nom d'utilisateurs *etc.*). Dans un premier temps, nous retirons toutes les options, identifiées comme commençant par '-'. Par exemple, les deux commandes successives `(ps,-aux)` ou bien `(ps,-ef)` sont toutes les deux considérées comme équivalentes et valant `ps`. On efface aussi tous les symboles génériques d'anonymisation. Les indicateurs de début et fin de sessions ne sont retirés quand dans un second temps car utilisés pour construire la matrice de transition.

4.2 Loi de Zipf

Comme décrit dans l'état de l'art, la loi de Zipf se valide par un ajustement dans un plan log-log rang fréquence. Le panneau de gauche de la Figure 1 illustre un ajustement de la loi de Zipf sur la première séquence du jeu de données. Comme attendu, les commandes de ce jeu de données suivent assez bien une distribution de Zipf ($s = 1.88$). Le coefficient de détermination ici est $R^2 = 0.94$ et servira de métrique pour évaluer l'ajustement.

4.3 Matrices de Transitions

Pour évaluer les dépendances entre commandes, on considère un modèle de Markov d'ordre 1, comme par exemple dans [31]. Si trois commandes X_1, X_2, X_3 se suivent, alors le modèle de Markov d'ordre 1 suppose que la fréquence d'apparition de X_3 ne dépend que de la commande X_2, mais pas de X_1. Autrement dit $p(X_3|X_1, X_2) = p(X_3|X_2)$. Cette probabilité conditionnelle est appelée la probabilité de transition. Pour caractériser les dépendances, on estime les probabilités de transitions pour toutes les paires de commandes. Une représentation de la matrice de transition associée à la première séquence du jeu de données est montrée sur le

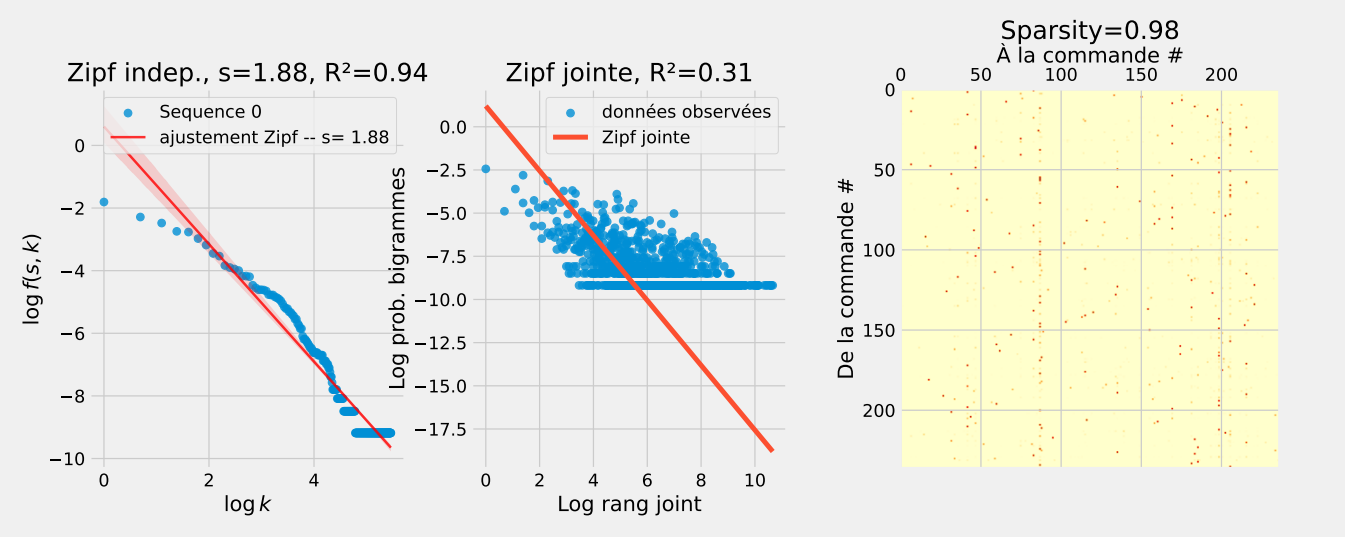

Figure 1: Plusieurs représentations des données Unix-Irvine pour la première séquence du jeu de données Unix-Irvine. Gauche : log des probabilités de chaque commande en fonction du log de son rang. L'ajustement de la loi de Zipf est donné ($s = 1.88, R = 0.94$). Milieu : log des probabilités de chaque bigramme en fonction de son rang joint. On montre aussi en rouge la loi théorique que devraient suivre les bigrammes si la séquence était zipfienne indépendante. On montre aussi l'ajustement à ce modèle ($R = 0.31$). Droite : Matrice de transition, où chaque pixel représente la probabilité de passer d'une commande à l'autre. Les pixels vont du rouge (1) au jaune (0). La proportion de probabilité de transition inférieures à 0.01 est donné par la *sparsity* (0.98).

panneau de droite de la Figure 1, où les couleurs vont de jaune (0) à rouge (1). A noter que pour une meilleure visibilité [6], l'ordre des commandes a été mélangé aléatoirement dans la matrice de transition, et dans tous le reste de l'article nous ferons la même chose. Nous introduisons ici la métrique de *sparsity* qui donne en pourcentage le nombre de probabilités de transitions nulles (en pratique, celles inférieures à 0.01). Les données réelles ont de nombreuses probabilités de transitions nulle, avec ici une sparsity de 0.98.

4.4 Bigrammes

Pour compléter la visualisation de la matrice de transition, nous proposons aussi une visualisation par bigramme. Un bigramme est une paire d'objets, ici de commandes. On s'intéresse plus précisément à la probabilité de chaque bigramme. Prenons deux commandes successives (X_i, X_{i+1}). Alors la probabilité de ce bigramme est $p(X_i, X_{i+1}) = p(X_{i+1}|X_i)p(X_i)$. Si les commandes suivent une loi de Zipf indépendante alors les bigrammes sont distribués selon une loi de Zipf de même paramètre qui utilise une autre définition du rang. En effet, avec k et k' le rang de X_i et X_{i+1} on a $p(X_{i+1}|X_i) = p(X_{i+1})$ (indépendance), $p(X_{i+1}) = \frac{C}{(k')^s}$ et $p(X_i) = \frac{C}{k^s}$, ce qui fait au final $p(X_i, X_{i+1}) = \frac{C^2}{(k' \times k)^s}$. Ainsi en posant $\tilde{k} = k' \times k$, le rang "joint", nous retrouvons une loi de Zipf sur les bigrammes de même pente :

$$\log p(X_i, X_{i+1}) = 2 \log C - s \log \tilde{k}, \tag{2}$$

qu'on appelle la loi de Zipf sur les bigrammes.

Pour des commandes qui suivent une loi de Zipf avec dépendances cette relation n'est plus vraie, et on s'en éloigne à proportion de la différence entre la probabilité conditionnelle $p(X_{i+1}|X_i)$ et la marginale $p(X_{i+1})$.

La probabilité des bigrammes, ainsi que l'ajustement théorique donné par l'Eq. (2) est donné sur le panneau du milieu de la Figure 1. Nous évaluons l'ajustement de ces données par rapport à la loi de Zipf sur les bigrammes grâce au coefficient de détermination [7], ici $R^2 = 0.31$, ce qui montre un mauvais ajustement. Cela implique que le modèle zipfien indépendant est mauvais, et que par conséquence il y a des dépendances entre les commandes.

5 UN ALGORITHME POUR GÉNÉRER DES SÉQUENCES ZIPFIENNES DÉPENDANTES

Comme évoqué dans l'introduction, il n'y a aujourd'hui à notre connaissance pas de méthode connue pour générer des séquences zipfiennes dépendantes. Nous traduisons certains aspects importants pour l'utilisateur de l'algorithme (*e.g.*, un chercheur, un concepteur d'étude) en propriétés désirables de la matrice de transition :

— L'utilisateur peut choisir le nombre de commandes, ce qui veut dire que la dimension de la matrice de transition doit être contrôlable.

6. Ce problème de visibilité se pose surtout dans le cas où l'on va comparer les matrices de transitions empiriques à celles d'une loi indépendante de Zipf.

7. Nous calculons le coefficient de détermination par rapport à la loi de Zipf sur les bigrammes, et non par rapport à la droite qui minimise l'erreur quadratique des bigrammes.

— Il ne doit pas y avoir de commande innaccessible, ce qui veut dire que la matrice est irréductible.
— Les commandes doivent être distributées suivant une loi de Zipf dont on contrôle le paramètre s, c'est à dire que la distribution stationnaire de la matrice de transition suit une loi de Zipf paramétrable par s.
— De nombreuses commandes ne doivent être executées qu'après certaines commandes, et de nombreuses commandes ne doivent presque jamais apparaître à la suite l'une de l'autre. Ceci implique que la matrice est creuse (*sparse*) : la plupart de ses entrées sont nulles.

Nous introduisons ensuite d'autres propriétés, de la matrice de transition qui sont utiles pour comprendre l'algorithme.

5.1 Propriétés de la Matrice de Transition

Propriété 1 (Distribution stationnaire). *La distribution stationnaire π_s de la chaîne de Markov (la fréquence des commandes une fois les conditions initiales passées) qui a pour matrice de transition P vérifie $P\pi_s = \pi_s$.*

Cette propriété permet de calculer les fréquences associées à chaque commande à partir de P, par un calcul de valeur propres (π_s est le vecteur propre associé à la valeur propre 1.)

Propriété 2. *La somme des termes de P sur une ligne somment à 1 : $\sum_i P_{i,j} = 1$*

Cette propriété découle de la définition d'une matrice de transition où la somme de toutes les probabilités doit être égale à 1.

Propriété 3. *La distribution stationnaire d'une loi de Zipf est décroissante : $\pi_1 > \pi_2 > \cdots > \pi_N$ avec $\pi_s = (\pi_1, \pi_2, \ldots, \pi_N)$*

Ceci découle directement de la définition de la loi de Zipf.

Propriété 4. *Les somme des termes de P sur une colonne vérifient $\sum_j P_{i,j}(\frac{i}{j})^s = 1$*

Ceci découle de la propriété 1 et de la définition de la loi de Zipf. En effet, on a quelquesoit i et j $\sum_j P_{i,j}\pi_j = \pi_i$, or $\pi_i = Ci^{-s}$, ce qui donne bien le résultat attendu.

5.2 Algorithme

Nous définissons d'abord l'écart Δ entre la distribution stationnaire π_s et celle de la loi de Zipf voulue p_z : $\Delta = \pi_s - p_z$. L'algorithme fonctionne ensuite de la sorte :

(1) Une matrice aléatoire irréductible [8] est initialisée avec pour moitié des termes une valeur de zéro. En pratique, on peut faire la proportion de zéros sans grand impact, sauf quand celle-ci se rapproche trop de 1.
(2) Cette matrice est "zipfianisée" et rendue valide, c'est à dire qu'on modifie les termes de la matrice de transition pour qu'elle réponde aux propriétés ci-dessus :
   (a) on change l'ordre des lignes pour respecter la propriété 3,
   (b) on normalise les colonnes pour vérifier la propriété 4, et
   (c) on normalise les lignes pour vérifier la propriété 2.
(3) La matrice zipfianisée est modifiée de sorte à ce que π_s tende vers p_z selon la mise à jour suivante :
   (a) Choisir au hasard un élément $P_{i,j}$ de la matrice de transition
   (b) Le modifier par : $P_{i,j} \leftarrow P_{i,j} \times A^{\Delta_j}$, où Δ_j est la $j$ième composante de Δ et où A est une constante supérieure à 1.
(4) Des dépendances sont injectées dans la matrice : On choisit au hasard un élément $P_{i,j}$ de la matrice de transition qu'on mets à 1, et ceci, de plus en plus souvent à mesure que la dimension de la matrice de transition augmente (pour N% des iterations, avec un maximum à 100%). [9] Il est nécessaire d'injecter des dépendances puisque sinon l'algorithme aura tendance à faire converger la matrice de transition vers une matrice de transition d'une loi zipfienne indépendante.
(5) Les étapes (2), (3) et (4) sont répétées jusqu'à convergence (pour $\|\Delta\|$ suffisament faible)

6 ÉVALUATION

Nous évaluons l'algorithme en calculant les métriques définies dans la section 4. Nous utilisons comme point de comparaison les données synthétiques zipfiennes indépendantes, et les séquences du jeu de données Unix-Irvine.

6.1 Données synthétiques indépendantes

Nous générons les données à partir d'une loi de Zipf indépendante ($s = 1.88$, comme pour la Figure 1) pour deux tailles de séquences différentes Figure 2 : le panneau du haut utilise la même taille d'échantillon (10808) et le même nombre de commandes (239) que pour la première séquence utilisée dans la Figure 1. Le panneau du bas reprend les même paramètres mais avec une taille d'échantillon de 1 million.

Nous observons notamment des bons ajustements pour la loi de Zipf sur les fréquences et sur les bigrammes (même si un moins bon ajustement est notable pour la loi de Zipf sur les bigrammes, en particulier pour la faible taille de séquence.) Nous observons aussi dans les deux cas des sparsity très proches de 1 *i.e.*, la plupart des probabilités de transition sont nulles. Graphiquement, des lignes verticales apparaissent dans la matrice de transition, qui correspondent aux commandes les plus fréquentes, vers lesquelles les autres commandes retournent [10].

6.2 Unix-Irvine

Comme second point de comparaison, nous utilisons le jeu de données Unix-Irvine, qui avait déjà été illustré pour un utilisateur avec la Figure 1, et dont l'évaluation sur l'ensemble des utilisateurs est résumée Figure 3. Nous illustrons aussi la loi de zipf sur les fréquences et bigrammes ainsi que les matrices de transitions pour

8. Pour vérifier le caractère irréductible de la matrice, on peut en pratique calculer P^n pour n très grand et voir si la matrice n'a pas de zéros. Une matrice avec des zéros placés aléatoirement sera très souvent irréductible tant qu'il n'y a pas trop de zéros — il suffit donc d'en générer aléatoirement et vérifier le caractère irréductible, et recommencer si elle ne l'était pas.

9. Il faut "zipfianniser" la matrice plus rapidement que l'on n'injecte des dépendances, ce qui nécessite dans des matrices de faible dimensions de ne pas trop ajouter des dépendances à la fois.

10. Nous comprenons maintenant l'intérêt de mélanger la matrice de transition pour la visualisation. Sans mélange, les lignes seraient collées à gauche puisque correspondant aux commandes les plus fréquentes, et relativement invisibles.

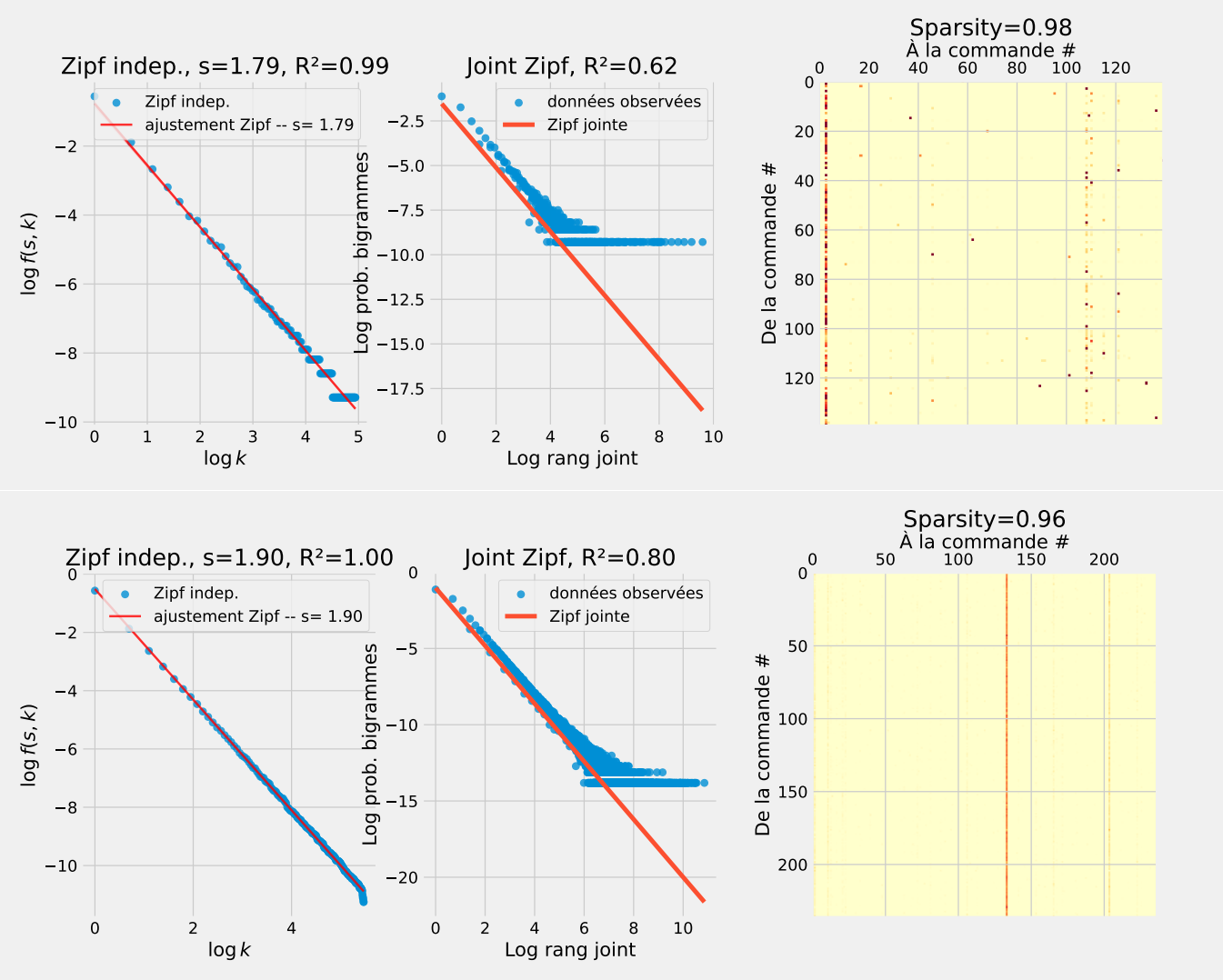

Figure 2: Visualisation et métriques identiques à la Figure 1 pour une séquence zipfienne indépendante. Haut : séquence longue de 10808 commandes. Bas : séquence longue de 1 million de commandes.

l'ensemble des utilisateurs en matériaux supplémentaires. Nous observons que :

- La plupart des probabilités de transition sont nulles (sparsity de 0.99 en moyenne), ce qui est proche de ce qu'on trouve avec la loi de Zipf indépendante.
- L'ajustement avec la loi de Zipf est bon (R^2 moyen de 0.95),
- Il y a assez peu de variabilité dans la valeur du paramètre s de la loi de Zipf : valeur de moyenne de 1.82 (min 1.65, max 1.94).
- L'ajustement avec la loi de Zipf sur les bigrammes est mauvais (R^2 moyen de 0.31).

La visualisation de la Figure 1 (et pour les autres séquences dans les matériaux supplémantaires) montre aussi que les bigrammes sont distribuées très différemment de celle que nous aurions eu sous l'hypothèse de séquence indépendante de Zipf, et nous ne retrouvons pas non plus les caractéristiques de lignes verticales dans la matrice de transition. Ceci implique que les commandes les plus fréquentes ne suivent pas n'importe quelle commande, et qu'il faut passer par d'autres commandes pour y retourner, ce qui met en évidence des dépendances fortes.

6.3 Performances de l'algorithme

6.3.1 Illustration. Le but de l'algorithme étant de générer des séquences zipfiennes avec dépendances, il faut vérifier que les séquences générées sont effectivement proches d'être zipfiennes et comportent des dépendances. En effet, la partie zipfianisation de l'algorithme peut avoir tendance à réduire les dépendances par normalisations successives, et dans une première version de l'algorithme la matrice de transition avait tendance a converger vers celle d'une zipfienne pure, problème que nous avons réglé par l'injection

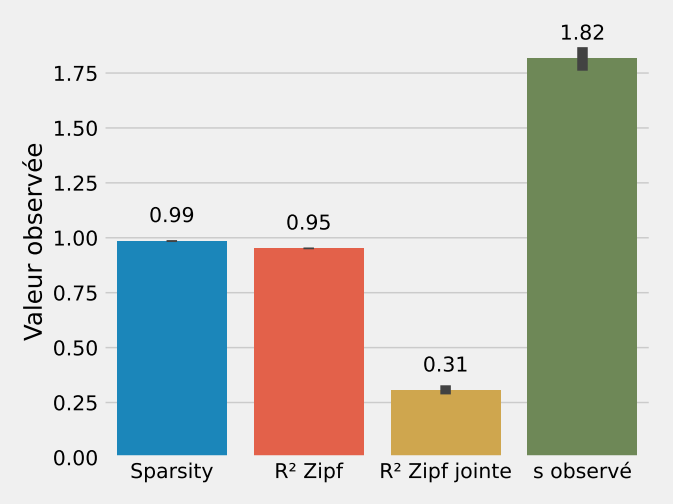

Figure 3: Évaluation du jeu de données Unix-Irvine sur les métriques décrites en section 4.

de dépendances (étape 4 de l'algorithme). Nous illustrons d'abord Figure 4 le résultat de l'algorithme, pour $s = 1.88$ et 239 commandes avec une séquence simulée de 10808 commandes (Figure 4, haut) pour comparaison avec la Figure 1 et 1 millions de commandes (Figure 4, bas).

Nous remarquons notamment que :

- que le caractère zipfien des données est bien conservé avec une certaine erreur sur le paramètre s [11]
- Une ressemblance assez forte avec les données Unix-Irvine au niveau des probabilités des bigrammes (voir Figure 1).
- une resemblance assez forte avec la matrice de transition des données Unix-Irvine, avec toutefois des lignes verticales plus marquées (comme pour les séquences zipfiennes indépendantes).

6.3.2 Évaluation. Pour cette évaluation, nous calculons une matrice de transition, l'utilisons pour générer une séquence de taille $N = 100000$ et ajustons les lois de zipf sur les fréquences et bigrammes puis calculons la *sparsity* observée, le paramètre s estimé, ainsi que les R^2 correspondant aux ajustements de zipf sur les fréquences et bigrammes. Ces métriques sont calculées pour une taille croissante du nombre de commandes différentes (de 10 à 150 commandes) et pour des valeurs de paramètre de la loi de Zipf s variables (de 1 à 2,5).

Les résultats sont présentés Figure 5. Nous observons notamment que :

- La sparsity reste très proche de 1 pour la plupart des conditions, sauf pour les nombres faible de commandes.
- Les ajustement des fréquences avec la loi de Zipf sont habituellements bons, même si ils ont tendance à diminuer avec les nombre de commandes. De la même manière, la valeur de s tend à s'éloigner de la consigne à mesure que le nombre de commandes augmente. Ceci est probablement en partie du au problème d'ajustement identifié dans la Figure 4 et décrit dans la note de bas de page numéro 8.
- L'ajustement par rapport à la loi de Zipf sur les bigrammes reste faible, ne dépassant que pour quelques cas $R = 0,5$, ce qui indique que l'algorithme permet bien de générer des séquences dépendantes.

Ainsi, l'algorithme se comporte comme nous le voulions, à l'exception de la valeur de s. Toutefois, ce problème n'est pas insurmontable. En effet, la Figure 5 n'est le résultat que d'une évaluation de l'algorithme pour chaque condition. Nous pouvons tout simplement relancer l'algorithme, ajuster le modèle de Zipf, et vérifier que le paramètre s est bien celui voulu.

7 DISCUSSION ET TRAVAUX FUTURS

7.1 Convergence de l'algorithme

Nous ne trouvons pas toujours une matrice valide au premier coup de l'algorithme. En effet, une matrice de transition avec $N = 239$ et une sparsity de 0.5 (valeur par défaut lors de l'initialisation) possède plus de 25000 paramètres à ajuster. De plus, nous n'avons démontré aucune guarantie quant à sa convergence : par exemple, certaines mises à jour peuvent faire augmenter la fonction coût. Pour des problèmes de cette dimension, le choix de la matrice initiale joue beaucoup sur les résultats. En pratique, pour l'évaluation, nous lançons 20 fois l'algorithme avec 10 itérations. Nous sélectionnons ensuite la meilleure matrice de transition que nous replaçons dans l'algorithme, pour un plus grand nombre d'itération, tout ceci de manière automatisée. L'ensemble prend entre quelques secondes et quelques dizaines de secondes suivant la valeur de N.

7.2 Hyperparamètres de l'algorithme

Les hyperparamètres sont des paramètres propres à l'algorithme et non au problème considéré. Dans notre algorithme, nous avons utilisé plusieurs hyperparamètres, que nous n'avons pas essayé d'optimiser :

- le facteur A qui conditionne la mise à jour. Plus on augmente ce chiffre, plus la convergence sera rapide mais on risque aussi de diverger plus souvent.
- le pourcentage d'itérations dans lesquels on injecte les dépendances : il faut injecter suffisamment de dépendances dans la matrice pour qu'elles aient un effet, l'étape (2) ayant tendance à affaiblir les dépendances. Toutefois, trop d'injections de dépendances et la matrice de transition ne donnera pas une distribution stationnaire zipfienne, l'étape (2) n'étant pas suffisante pour la corriger. Il faut donc un taux d'injection qui dépend de la dimension N de la matrice de transition. Nous avons choisis un taux linéaire (en N), mais peut-être qu'un taux quadratique (en N^2, et donc linéaire en nombre d'éléments de la matrice) serait plus pertinent.
- On peut aussi jouer sur la valeur des dépendances qu'on injecte. Ici, on injecte toujours 1 dans la matrice de transition à l'étape (4). Toutefois, le 1 ne permet pas nécessairement d'imposer une probabilité de transition forte, puisque les étapes de normalisation et de zipfiannisation vont diluer ce chiffre

11. En fait, cette erreur est probablement due à la méthode d'estimation utilisée. En effet, on a effectué une régression linéaire simple, cette dernière accordant autant de poids à tous les points. Or, on devrait logiquement donner un poids plus fort aux points de bas rang dont la fréquence est nettement supérieure. Probablement qu'une estimation au moindre carrés pondérés donnerait une estimation plus fiable ; dans ce cas un s plus faible.

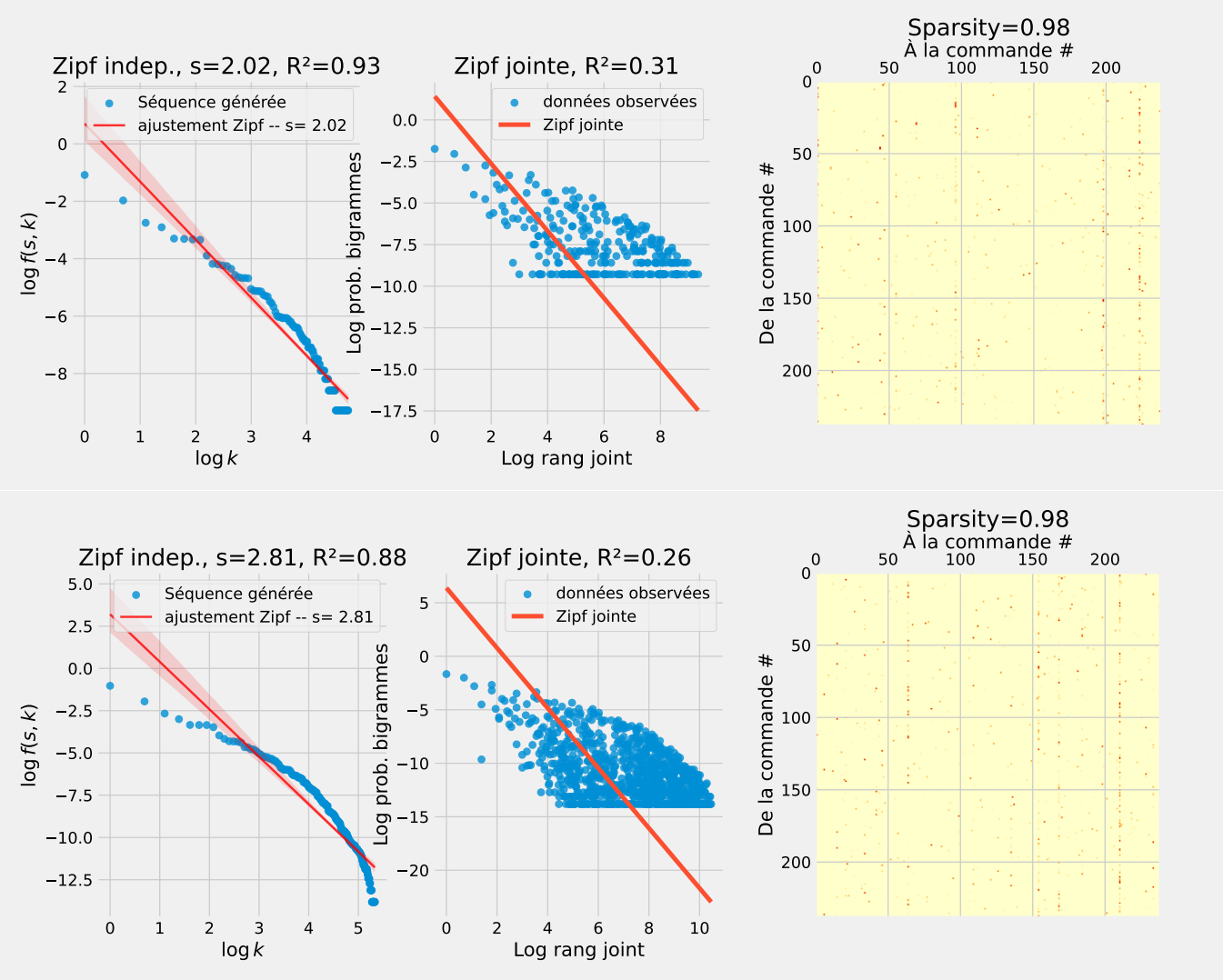

Figure 4: Performances de l'algorithme pour deux tailles de séquences différentes (10808 en haut, $1e^6$ en bas), pour $s = 1.88$ et $N = 239$. Les visualisations sont identiques à celles de la Figure 1.

(concrètement pour imposer une probabilité de transition égale à 1 après normalisation il faudrait mettre une valeur très grande, disons $1e^6$). Jouer sur ce paramètre pourrait potentiellement améliorer la correspondance avec les matrices de transition empiriques qui semblent composées plus souvent de probabilités de transition élevées (comparer le nombre de points rouges hors verticales dans les Figure 1 et Figure 5.)

7.3 Estimation des bigrammes

Les commandes suivant une loi de Zipf, les 10 commandes les plus fréquentes représentent plus de 98% des commandes. Au contraire, la plupart des commandes ont des fréquences d'apparition très faibles, ce qui rend difficile leur observation, et donc leur estimation fiable. Par exemple, il y a plus de 50000 paramètres à estimer dans la matrice de transition de taille 239×239 de la Figure 1, alors que la séquence ne fait que 10000 commandes. Un travail serait à faire pour améliorer l'estimation de ces matrices de transition ; réduire le nombre de paramètres passera probablement par un ajout de certaines hypothèses sur les séquences.

7.4 Évaluation de l'algorithme

Nous avons évaluer l'algorithme à travers la Figure 5. Cette évaluation est utile pour au moins deux raisons. Premièrement, elle montre que l'algorithme fonctionne, et comment il se comporte pour des valeurs crédibles. Deuxièmement, il permet d'identifier plusieurs axes d'améliorations. Le premier est que la valeur du paramètre s de la séquence générée ne correspond pas toujours à celle demandée. On a vu qu'une raison était l'estimation de s qui n'était pas forcément fiable. Un travail supplémentaire devra être fait de ce côté. Une fois l'algorithme d'estimation mieux caractérisé, il faudra

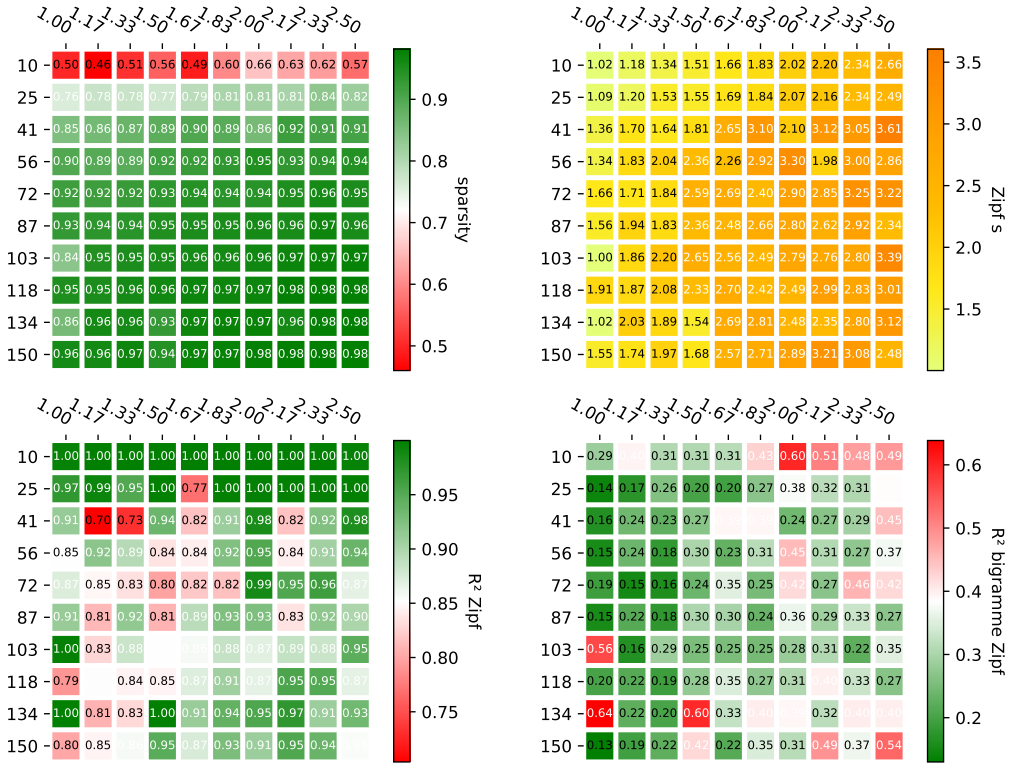

Figure 5: Évaluation de l'algorithme sur les critères de sparsity (en haut à gauche), de valeur de s estimé (en haut à droite), et R^2 sur les lois de Zipf sur fréquences et bigrammes (en bas à gauche et droite respectivement). Métriques évaluées avec une seule exécution de l'algorithme et pour des séquences de taille $N = 100000$, pour s allant de 1 à 2,5 et pour un nombre de commandes allant de 10 à 150

voir si en effet l'algorithme réussit à garantir cette valeur. Il faudra aussi trouver d'autres métriques pour caractériser la ressemblance entre les séquences empiriques et celles générées par l'algorithme, en particulier pour vérifier qu'on retrouve bien des séquences comparables entre celles empiriques et générées par l'algorithme. Enfin, il pourrait être judicieux d'évaluer l'effet des diverses séquences (réalistes, générées avec ou sans dépendances) en repliquant une expérience connue, comme par exemple celle de Findlater *et al.* [8].

7.5 Vers des chaînes de Markov d'ordre plus élevés

Comme décrit sous-section 2.2, certaines dépendances semblent s'étaler sur plusieurs commandes. Des méthodes statistiques existent pour caractériser l'ordre d'une chaîne de Markov [17], qu'il faudrait tester sur des jeux de données de commandes observés lors d'usage naturel. Une seconde question est alors de savoir si l'on peut généraliser l'algorithme pour des chaines de Markov d'ordre plus élevé. La méthode présentée ici s'appuie sur des matrices de transition qui représentent en fait les probabilités conditionnelles $p(X_{i+1}|X_i)$ de passer de la commande X_i à X_{i+1}. On peut tout à fait considérer la probabilité conditionnelle $p(X_{i+2}|X_{i+1}, X_i)$ pour traîter le cas de la chaîne de Markov d'ordre 2. Seulement, il y a un coup à payer au niveau de la matrice de transition, qui devient un objet de dimension 3, avec N^3 probabilités à estimer. De manière générale, pour générer une chaine de Markov d'ordre n avec l'algorithme décrit ici il faudra estimer au préalable N^{n+1} probabilités, la plupart étant extrèmement rares. Il va sans dire que dans l'état actuel de l'algorithme, ceci n'est réalisable que pour un nombre N relativement faible de commandes différentes.

RÉFÉRENCES

[1] Samarth Aggarwal, Rohin Garg, Abhilasha Sancheti, Bhanu Prakash Reddy Guda, and Iftikhar Ahamath Burhanuddin. 2020. Goal-driven command recommendations for analysts. In *Proceedings of the 14th ACM Conference on Recommender Systems*. 160–169.

[2] Gilles Bailly, Antti Oulasvirta, Duncan P. Brumby, and Andrew Howes. 2014. Model of Visual Search and Selection Time in Linear Menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. ACM, New York, NY, USA, 3865–3874. https://doi.org/10.1145/2556288.2557093

[3] Donald T Campbell and Julian C Stanley. 2015. *Experimental and quasi-experimental designs for research.* Ravenio books.

[4] Andy Cockburn and Carl Gutwin. 2009. A predictive model of human performance with scrolling and hierarchical lists. *Human–Computer Interaction* 24, 3 (2009), 273–314.

[5] Matthieu Cristelli, Michael Batty, and Luciano Pietronero. 2012. There is more than a power law in Zipf. *Scientific reports* 2, 1 (2012), 812.

[6] John J. Darragh, Ian H. Witten, and Mark L. James. 1990. The reactive keyboard : A predictive typing aid. *Computer* 23, 11 (1990), 41–49.

[7] Ben Derrick, Deirdre Toher, and Paul White. 2016. Why Welch's test is Type I error robust. *The quantitative methods for Psychology* 12, 1 (2016), 30–38.

[8] Leah Findlater and Joanna McGrenere. 2004. A comparison of static, adaptive, and adaptable menus. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 89–96.

[9] Bruno Fruchard, Eric Lecolinet, and Olivier Chapuis. 2018. Impact of semantic aids on command memorization for on-body interaction and directional gestures. In *Proceedings of the 2018 International Conference on Advanced Visual Interfaces*. 1–9.

[10] K George. 1935. Zipf. The Psychobiology of Language. *An Introduction to Dynamic Philology* (1935).

[11] Saul Greenberg and Ian H Witten. 1985. Adaptive personalized interfaces—A question of viability. *Behaviour & Information Technology* 4, 1 (1985), 31–45.

[12] Saul Greenberg and Ian H Witten. 1993. Supporting command reuse : empirical foundations and principles. *International Journal of Man-Machine Studies* 39, 3 (1993), 353–390.

[13] Tovi Grossman, Pierre Dragicevic, and Ravin Balakrishnan. 2007. Strategies for accelerating on-line learning of hotkeys. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1591–1600.

[14] Stephen José Hanson, Robert E Kraut, and James M Farber. 1984. Interface design and multivariate analysis of UNIX command use. *ACM Transactions on Information Systems (TOIS)* 2, 1 (1984), 42–57.

[15] Melanie Hartmann and Daniel Schreiber. 2007. Prediction Algorithms for User Actions.. In *LWA*. 349–354.

[16] Wen-Hua Ju and Yehuda Vardi. 2001. A hybrid high-order Markov chain model for computer intrusion detection. *Journal of Computational and Graphical Statistics* 10, 2 (2001), 277–295.

[17] Richard W Katz. 1981. On some criteria for estimating the order of a Markov chain. *Technometrics* 23, 3 (1981), 243–249.

[18] Judy Kay and Richard C Thomas. 1995. Studying long-term system use. *Commun. ACM* 38, 7 (1995), 61–69.

[19] Gordon Kurtenbach and William Buxton. 1993. The limits of expert performance using hierarchic marking menus. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*. 482–487.

[20] Benjamin Lafreniere, Andrea Bunt, John S Whissell, Charles LA Clarke, and Michael Terry. 2010. Characterizing large-scale use of a direct manipulation application in the wild. In *Proceedings of Graphics Interface 2010*. 11–18.

[21] Terran Lane. [n.d.]. UNIX User Data. UCI Machine Learning Repository. DOI : https ://doi.org/10.24432/C5302K.

[22] Frank Linton, Deborah Joy, and Hans-Peter Schaefer. 1999. Building user and expert models by long-term observation of application usage. In *UM99 User Modeling : Proceedings of the Seventh International Conference*. Springer, 129–138.

[23] Wanyu Liu, Gilles Bailly, and Andrew Howes. 2017. Effects of frequency distribution on linear menu performance. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 1307–1312.

[24] I Scott MacKenzie and R William Soukoreff. 2003. Phrase sets for evaluating text entry techniques. In *CHI'03 extended abstracts on Human factors in computing systems*. 754–755.

[25] Justin Matejka, Wei Li, Tovi Grossman, and George Fitzmaurice. 2009. CommunityCommands : command recommendations for software applications. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. 193–202.

[26] Jaap MJ Murre and Joeri Dros. 2015. Replication and analysis of Ebbinghaus' forgetting curve. *PloS one* 10, 7 (2015), e0120644.

[27] Andrew Sears and Ben Shneiderman. 1994. Split menus : effectively using selection frequency to organize menus. *ACM Transactions on Computer-Human Interaction (TOCHI)* 1, 1 (1994), 27–51.

[28] Claude Elwood Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal* 27, 3 (1948), 379–423.

[29] AG Sutcliffe and AC Old. 1987. Do users know they have user models? Some experiences in the practice of user modelling. In *Human–Computer Interaction–INTERACT'87*. Elsevier, 35–41.

[30] Kashyap Todi, Gilles Bailly, Luis Leiva, and Antti Oulasvirta. 2021. Adapting user interfaces with model-based reinforcement learning. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–13.

[31] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. 1999. Detecting intrusions using system calls : Alternative data models. In *Proceedings of the 1999 IEEE symposium on security and privacy (Cat. No. 99CB36344)*. IEEE, 133–145.

[32] Thomas D Wickens and Geoffrey Keppel. 2004. *Design and analysis : A researcher's handbook*. Pearson Prentice-Hall Upper Saddle River, NJ.

[33] Longqi Yang, Chen Fang, Hailin Jin, Matthew D Hoffman, and Deborah Estrin. 2017. Personalizing software and web services by integrating unstructured application usage traces. In *Proceedings of the 26th International Conference on World Wide Web Companion*. 485–493.