UNIVERSITÉ DE SHERBROOKE
DÉPARTEMENT D'INFORMATIQUE

Rapport de fin de session

Dans le cadre du cours:
IFT593 - Projet en systèmes intelligents

Travail présenté
à
Pr Pierre-Marc Jodoin

Par
Jeremi Levesque - LEVJ1404

15 décembre 2023

https://github.com/MrZarfir/TrackToLearn
Branche *jeremi/to/reward*

Table des matières

1 Introduction

Le projet du connectome humain est un défi important dans le domaine des neurosciences dont l'objectif est de reconstituer l'ensemble des connexions du cerveau. La tractographie, qui fait usage de l'imagerie de diffusion, est la seule méthode non-invasive qui permet de reconstruire informatiquement la connectivité structurale du cerveau et de certains autres tissus fibreux. Cette technique qui connaît un intérêt croissant dans la recherche est aussi utilisée dans certains cas cliniques. La tractographie est utilisée comme un outil important pour la recherche au niveau du cerveau, mais aussi pour planifier et conduire des chirurgies d'épilepsie ou de tumeurs au cerveau au centre de recherche du CHUM. Les axones (fibres de matière blanche) du cerveau sont particulièrement intéressantes à utiliser puisqu'elles forment plusieurs réseaux électriques dans le cerveau pour y communiquer de l'information. La propagation de l'information dans le cerveau n'est pas aléatoire, les chemins de d'acheminement d'information entre les neurones du cerveau sont bien définis.

Une image IRM capture en trois ou quatre dimensions l'information anatomique d'un patient sous une certaine résolution. Cette résolution est clairement définie par l'instrument de capture et elle est d'une valeur finie en trois dimensions: on parle d'un voxel. Un voxel est donc un volume de capture qui possède des informations relatives à la région qu'il englobe. Par exemple, pour l'IRM de diffusion, on retrouve dans chaque voxel des informations sur la direction et l'intensité de propagation de l'eau dans le cerveau. En revanche, les axones que la tractographie cherche à modéliser sont des structures très petites ayant un diamètre de seulement 1 à 15 μm[1], alors que les voxels qui permettent d'imager le cerveau d'un individu sont généralement de taille de quelques millimètres (ex: 2mm x 2mm x 2.6mm ou 1mm x 1mm x 1mm). Il est donc pratiquement impossible de capturer de l'information sur chaque fibre d'axone de manière individuelle puisque chaque voxel mesure une grande quantité de fibres qui s'entrecroisent possiblement. Ainsi, il devient relativement complexe de reconstituer parfaitement la structure interne du cerveau d'un individu si on ne possède pas l'information sur chaque fibre de manière individuelle et précise. Comme on sait que le cerveau est composé à 90% d'eau, l'IRM de diffusion est donc un processus d'imagerie complexe qui permet de quantifier la diffusion de l'eau à l'intérieur du cerveau. Dans un espace libre, l'eau se diffuse continuellement dans toutes les directions de manière uniforme. Cependant, lorsque l'eau est contrainte dans un certain espace, alors la direction de la diffusion de l'eau en est affectée. Donc, comme les axones forment des sortes de tubes allongés, l'eau qui forme les axones est contrainte de se diffuser dans la même direction que la structure de la fibre. Donc, si l'eau se déplace le long de l'axone, on a seulement à reconstituer la route empruntée par l'eau pour reconstruire les fibres. Par analogie, c'est comme si on observait seulement le déplacement des véhicules sur la route afin de reconstruire les autoroutes et les routes.

La modélisation de la diffusion de l'eau dans chacun des voxels peut être approximée par plusieurs différents types de modèles (tenseurs, fonction de distribution de l'orientation d'une fibre, etc.)[2]. Ces modèles nous permettent donc d'estimer la direction de la propagation de l'eau et par le fait même, d'estimer l'orientation des fibres en fonction des différents voxels. Avec ce genre d'information, au niveau le plus simple possible, on peut lancer des algorithmes de tractographie déterministes simples qui ne font que suivre les directions principales de la diffusion de l'eau dans chaque voxel pour y reconstruire globalement une structure qui semble plausible. En termes simples, nous savons qu'une fibre a une longueur maximale, qu'elle ne peut pas changer de direction de manière trop abrupte et

[1] https://fr.wikipedia.org/wiki/Axone

[2] Voir le cours de Maxime Descoteaux: https://drive.google.com/file/d/1xrn1NYsFfVCnP0xvlnfxAasbwCpy0krx/view

qu'elle doit commencer et terminer dans la matière grise. Une fibre qui sort de la matière blanche ne peut pas par exemple passer à travers les ventricules du cerveau. Une fibre doit forcément connecter deux sections de matière grise. Plusieurs méthodes et techniques avancées permettent de retirer d'autres types de fibres qui ne sont pas anatomiquement possibles afin de tenter de retirer le plus possible de fausses fibres. En revanche, toutes les méthodes les plus récentes qui essaient de résoudre ce problème souffrent quand même d'importantes quantités de fibres anatomiquement non plausibles.

Comme les réseaux de neurones artificiels connaissent une résurgence particulièrement importante depuis quelques années dans tous les domaines, il a été proposé d'essayer d'utiliser les méthodes d'apprentissage machine afin de possiblement améliorer les résultats des algorithmes de tractographie classique. Les algorithmes de tractographie semblent lentement plafonner en terme d'amélioration sur leur performances, donc les méthodes de *Machine Learning* pourraient contribuer à améliorer ces résultats. Le premier problème qu'on rencontre est que les méthodes d'apprentissage machine sont en majorité dépendantes d'une grande quantité de données **annotées**. Afin d'annoter les différentes régions de matière blanche dans un cerveau, il faut une bonne période de temps afin qu'un expert (i.e. un neuroscientifique) puisse correctement délimiter ces régions. Aussi, si plusieurs experts annotent un même cerveau, le résultat ne sera pas nécessairement le même. Il y a donc une variabilité inter-experts en plus que certains faisceaux de fibres varient d'un patient à l'autre. Ce n'est donc pas une approche réaliste si on veut générer un immense ensemble de données qui permettra à des méthodes d'apprentissage machine d'apprendre. De plus, les données annotées seraient limitées au savoir des experts qui, bien qu'ils soient très compétents, ne connaissent pas nécessairement des faisceaux qui sont encore potentiellement inconnus par la communauté. Les méthodes de *Machine Learning* essaient d'apprendre les données annotées, mais ne pourraient pas extrapoler afin de découvrir de nouvelles structures inconnues [1].

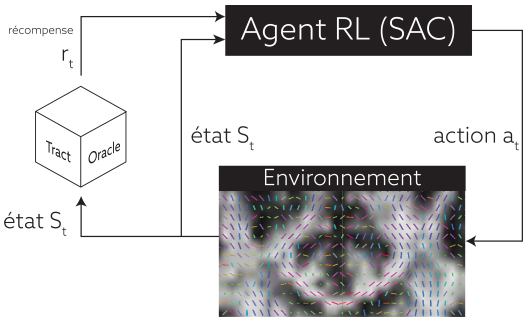
C'est donc où les méthodes d'apprentissage par renforcement (AR) entre en jeu afin d'espérer contourner la contrainte d'annotations des données. Au lieu de nécessiter des données annotées, un agent qui apprend par renforcement nécessite *seulement* la définition d'un comportement qu'il doit adopter.

Une bonne mise en contexte est effectuée avant de mettre en place les objectifs et les livrables du projet. Ceux-ci sont détaillés à la section 3.

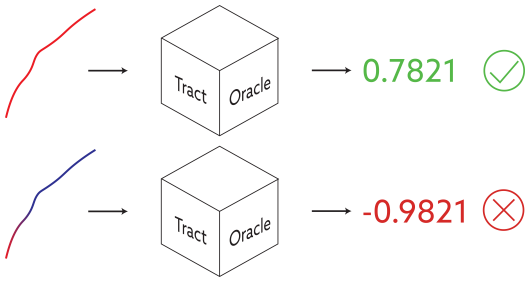
2 Travaux reliés

2.1 TrackToLearn

Ainsi, avec une simple définition du comportement qu'il doit adopter, l'agent explore donc l'environnement par lui-même. Ce comportement est défini à travers une fonction de récompense qui lui attribue des récompenses élevées lorsqu'il se retrouve dans un état désiré et des récompenses plus faibles et même négatives dans le cas contraire. Les méthodes d'AR sont particulièrement utilisées dans des contextes de jeux comme Atari [2], Dota [3] ou encore StarCraft II [4] et sont même plus récemment utilisées pour perfectionner des modèles de langues comme ChatGPT [5]. Antoine Théberge, lui, posera le problème d'apprentissage par renforcement dans le contexte de la tractographie en 2021 en développant le module TrackToLearn en Python [6]. Ce module a permis d'évaluer la performance de plusieurs algorithmes d'apprentissage par renforcement qui ont démontrés de solides performances dans des environnements complexes. En créant un tractogramme de manière déterministe, on le construirait en suivant les directions principales qui indiquent une



(a) Architecture générale de TrackToLearn lors de l'inférence. L'agent propage les streamlines en regardant les effets sur la validité de la fibre courante.



(b) Utilisation du TractOracle

Figure 1: Environnement de *tracking* basé sur des techniques d'apprentissage par renforcement.

diffusion de l'eau plus importante. Dans cet environnement, notre agent est une entité logicielle qui apprend à parcourir par lui même la connectivité du cerveau. Plus formellement, l'état s_t dans lequel se retrouve un agent est défini comme étant le tracé courant de la fibre, avec sa position spatiale ainsi que les directions de la diffusion de l'eau dans cet état. L'agent choisit l'action a_t, définie comme étant un pas dans une certaine direction lors de la tractographie, qui maximise les récompenses qu'il obtient en arrivant dans des états que notre définition du comportement juge adéquat. Au final, les résultats produits étaient compétitifs par rapport aux autres approches utilisées classiquement. L'algorithme évalué comme étant le plus performant dans ce contexte est celui de Soft Actor-Critic [7].

Il réalise rapidement qu'avec ce genre de méthodes, la qualité et la fiabilité des tractogrammes dépendent directement de la modélisation de la récompense attribuée à l'agent lors de son exploration. Dans cette recherche, il a été suggéré que l'évaluation de la qualité des fibres (ou streamlines) était peut-être trop simpliste. Ainsi, on propose donc une fonction de récompense (qui évalue la qualité des fibres) basée sur des algorithmes de classification et de validation des voies axonales (i.e. FINTA [8], Extractor [9]). Cette fonction, nommée TractOracle [10], est un réseau de neurones qui approxime la qualité de chaque fibre en lui attribuant un score. L'objectif de son travail est de pouvoir arrêter et rejeter des streamlines qui semblent prendre une direction incohérente avec ce qui est connu en l'anatomie afin de diminuer le taux de faux positifs dans les méthodes courantes. Le TractOracle jouera une importance cruciale dans la performance de l'algorithme implémenté à la section 4 puisqu'il est utilisé comme un outil central de ce projet.

2.2 Particle Filtering Tractography

La tractographie par filtrage de particules (PFT) [11] est une technique qui permet de réduire le nombre de streamlines qui s'arrêtent de façon prématurée à l'extérieur de la matière blanche. Évidemment, comme l'objectif de tracer les fibres de matière blanche est de connecter des régions de matière grise, l'algorithme ne considère pas une streamline qui termine en bordure de la matière grise comme une streamline qui a terminée prématurément puisqu'elle a atteint l'objectif. PFT vient s'initier juste avant de faire un pas qui terminerait la streamline de façon prématurée afin de forcer la streamline a continuer son tracé à l'intérieur de la matière blanche. On dénote $\delta_b$mm

comme la distance de *backtracking* (retour en arrière) et $\delta_f$mm comme la distance à propager à l'avant. L'objectif est d'estimer une streamline streamline initialisée à une distance de δ_b mm avant l'événement d'arrêt prématuré et de propager cette streamline vers l'avant de δ_f mm plus loin pour voir si l'événement d'arrêt prématuré est résolu. Si la streamline atteint la matière grise, elle est stoppée, mais conservée comme étant une streamline valide alors que si la streamline ne s'arrête pas et continue son parcours dans la matière blanche, alors le *tracking* de cette fibre continue normalement jusqu'à ce qu'elle soit encore arrêtée par un critère d'arrêt. PFT utilise un ensemble de N particules qui sont associés à des poids qui varie selon leur validité. À partir de la position initiale de $\delta_b$mm avant l'événement d'arrêt original, l'algorithme lance plusieurs parcours de streamlines probabilistes de sorte à explorer les avenues possibles pour la streamline en pénalisant (réduisant les poids respectifs) les streamlines qui se propagent à l'extérieur de la région d'intérêt.

3 Objectifs

Dans la même lignée que les recherches présentées ci-haut, notre objectif dans ce projet est d'explorer une méthode pour augmenter le taux de streamlines qui sont valides et par le fait même de réduire le taux de streamlines invalides. L'idée pour ce projet est d'explorer une combinaison entre la tractographie par filtre de particules (section 2.2) et l'approche avec un Oracle qui permet d'évaluer la qualité des streamlines. L'algorithme PFT présenté ci-dessus explore plusieurs chemins en parallèle et la streamline résultante est tirée d'une distribution des streamlines résultantes qui sont pondérées avec certains poids selon la validité des streamlines. Dans notre cas, plutôt que d'estimer une distribution des streamlines, on veut implémenter une approche qui utilise l'oracle afin d'y déterminer la meilleure streamline.

On ne s'attend pas nécessairement à des résultats conclusifs, mais on cherche à avoir une preuve de concept pour évaluer le potentiel d'une telle approche avec une approximation de la qualité des fibres. Cette preuve de concept devra avoir des visualisations appropriées pour confirmer l'implémentation de l'algorithme, mais aussi pour évaluer les prédictions faites par l'oracle. Le code sera implémenté directement à l'intérieur du module TrackToLearn dans une branche séparée sur GitHub.

4 Méthodologie

Comme mentionné plus haut, la majorité de ce projet se concentre à tenter d'améliorer les résultats d'une approche par apprentissage par renforcement. Donc, l'entièreté du développement est réalisé à l'intérieur même du module TrackToLearn qui est disponible publiquement. Pour mon implémentation qui n'est qu'une preuve de concept à ce stade, le travail a été réalisé dans une branche distincte, jeremi/to/reward en se basant sur la branche to/reward du dépôt principal. Une bonne partie du projet nécessitait que mon code cohabite avec certaines utilisations de PyTorch, pour l'utilisation des modèles et des différents agents, ainsi que DiPy pour la gestion des tractogrammes produits. Afin de reproduire un algorithme similaire à PFT, on doit pouvoir intercepter les streamlines juste avant qu'elles ne soient stoppées de façon prématurée. L'environnement *tracking_env* était l'emplacement idéal où introduire un *backtracking* des streamlines qui s'arrêtent puisqu'on teste les conditions d'arrêt dans la fonction *step()*. Dans la preuve de concept présentée ici, on utilise le terme *rollout* pour définir ce qui est une particule dans le contexte de PFT: un rollout est une continuation de la streamline qui s'arrête et qui est utilisée pour potentiellement trouver une meilleure direction pour la fibre originale en question.

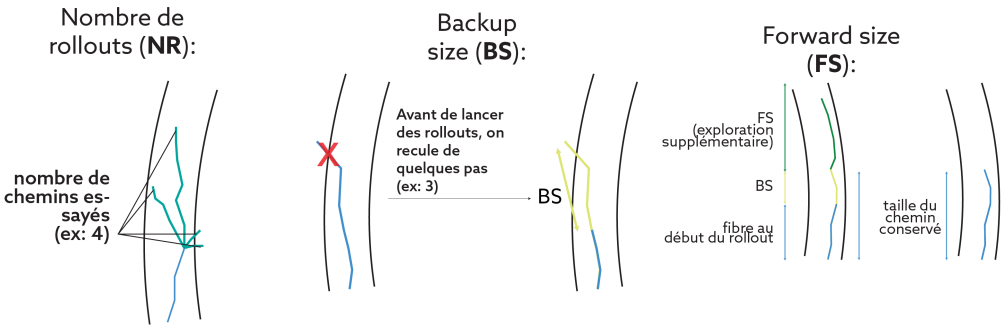


Figure 2: Illustration de l'usage des paramètres pour l'algorithme de *backtracking*.

Notre algorithme de *backtracking* et de rollout des streamlines invalides est englobé dans l'architecture courante du projet comme étant simplement un nouvel environnement nommé *rollout_env* (l'environnement de rollouts). Cet environnement est nécessite de faire des pas et d'évaluer les conditions d'arrêts des streamlines comme le fait BaseEnv et ils sont donc similaires en ce sens. Dorénavant, comme on pourrait vouloir offrir cette optimisation à la PFT à tous les environnements possibles, chaque environnement qui dérive de BaseEnv contiendra aussi un environnement de rollouts. Ainsi, dans la fonction step() de l'environnement de tracking, on vient appeler notre fonction *rollout()* lorsqu'il y a des streamlines qui s'arrêtent durant l'exécution et si cet algorithme permet de trouver de meilleurs chemins de matière blanche, on les met à jour et on continue le tracking pour les streamlines qui sont toujours actives.

Tout d'abord, on vient ajouter un critère d'arrêt qui permet de différencier les fibres qui s'arrêtent dans la matière grise (STOPPING_TARGET) des autres fibres. En faisant cela, il est donc possible dans la première étape d'exécution de notre algorithme de filtrer ces fibres de sorte à ne pas les modifier puisqu'elles atteignent l'objectif attendu. Les paramètres personnalisables de l'algorithme proposés sont définis de la manière suivante (une visualisation de ces paramètres est aussi disponible grâce à la figure 2):

- *Backup size*, BS (équivalent à δ_b): nombre de pas de la taille du retour en arrière lors du backtracking d'une fibre qui s'arrête prématurément.
- *Forward size*, FS (similaire à δ_f): nombre de pas supplémentaires effectuées pour aller explorer le potentiel d'un rollout dans les prochaines étapes. Par exemple, si la fibre originale était formée de 15 pas et que la FS est de 5, alors on va effectuer le backtracking de BS pas vers l'arrière, et le rollout sera déroulé jusqu'à atteindre 20 pas.
- Nombre de rollouts (NR) (similaire à N): nombre de rollouts testés en même temps pour chaque streamline.

L'approche utilisée dans ce projet se résume bien en 5 étapes principales qui sont bien illustrés dans la figure 3.

1. Dès la première étape, on vient identifier seulement les fibres qui sortent de la matière blanche. Notamment celles qui se tracent dans le liquide céphalo-rachidien ou encore juste complètement à l'extérieur du masque de matière blanche ce qui est invalide.

2. Pour chaque streamline identifiée, on revient de BS pas en arrière (paramètre ci-haut) afin d'avoir possiblement avoir plus de jeu avant d'atteindre la zone concernée par le critère d'arrêt. Si la longueur courante de la streamline est trop petite pour pouvoir *backtrack*, aucun rollout n'est fait pour éviter de couper la position initiale de la streamline. D'ailleurs, l'implémentation courante empêche un backtracking trop près du point initial de la streamline de sorte que ce point initial ne soit pas remplacé. Si on remplaçait la *seed* initiale, il nous serait impossible de trouver une direction dans laquelle propager notre streamline puisque nous n'aurions plus de position de référence.

3. Ensuite, on utilise l'agent SAC pré-entraîné afin de générer NR streamlines différentes. La sortie de la politique d'un agent SAC nous permet de contrôler la stochasticité des actions qu'il effectue. Ainsi, dans ce cas, on vient utiliser 1.1 fois la variance prédite pour une action prédite de sorte à avoir un comportement un peu plus aléatoire. Sans ça, l'agent n'explorerait pas un grand éventail de chemins possibles à partir de la streamline courante. Une bonne stochasticité nous permettra d'échantillonner plus adéquatement l'éventail des possibilités des directions qui semblent bonnes selon l'agent. De plus, en ajustant le paramètre $FS > 0$, on vient explorer propager ces fibres intermédiaires en faisant plus de pas que nécessaires. Par exemple, si la longueur de la streamline qui s'arrêtre prématurément est de 8 pas de long et que $FS = 3$, alors on va propager notre ensemble de sections de streamlines jusqu'à ce qu'à atteindre une longueur totale de 11 $(8 + FS)$

4. Ce qui est particulièrement utile avec le TractOracle présenté ci-dessus, c'est que maintenant, il ne suffit que de l'utiliser en lui donnant en entrée l'entièreté des fibres qui sont produites afin d'y évaluer leur score. On vient donc seulement conserver la fibre qui maximise le score prédit par l'oracle. Dans le cas où toutes les streamlines ne parviennent pas à une amélioration de l'originale ou si la streamline avec le score le plus grand est plus courte que l'originale, alors on ne corrige simplement pas cette fibre: on arrête donc de propager cette fibre.

5. Comme on explore FS pas plus loin que la taille originale, on vient, dans cette dernière étape, uniquement conserver la taille originale de la streamline. On pourrait vouloir conserver la nouvelle streamline au complet. Cependant, on cherche à aller voir plus loin uniquement pour avoir une meilleure idée de la qualité et la validité d'une direction. C'est donc pour avoir un score plus représentatif et si cette trajectoire était effectivement la meilleure, alors l'agent devrait normalement la retrouver par la suite, ce qui ajoute un peu de redondance dans l'algorithme pour confirmer que le score qui était donné était raisonnable. De plus, c'est aussi ça résulte en une implémentation plus simple qui, à l'extérieur de l'environnement de *rollout_env*, ne demande aucune modification. Sans ça, il faudrait modifier l'algorithme courant de sorte à pouvoir propager des streamlines de taille différentes en même temps (TrackToLearn propage généralement environ 50 000 fibres en même temps). Avec un traitement uniforme, on peut avoir une implémentation plus simple et uniforme à travers le programme.

Ainsi, l'implémentation de cet algorithme est vraiment abstrait de l'environnement qui l'utilise. Il ne suffit que de donner quelques informations lors de l'initialisation du *rollout_env* et il est ainsi possible d'utiliser cet environnement afin qu'il puisse venir possiblement corriger les streamlines attendues. Son utilisation est donc simple à intégrer dans différents environnements. Comme ce projet sera probablement adapté et complété durant ma maîtrise qui débute en Janvier 2024, cet algorithme pourrait être intégré notamment durant l'entraînement d'un agent dans sa boucle d'apprentissage par renforcement pour plusieurs agents différents, puisqu'il ne faudrait que faire appel aux fonctionnalités de l'environnement développé pour avoir une correction "automatique"

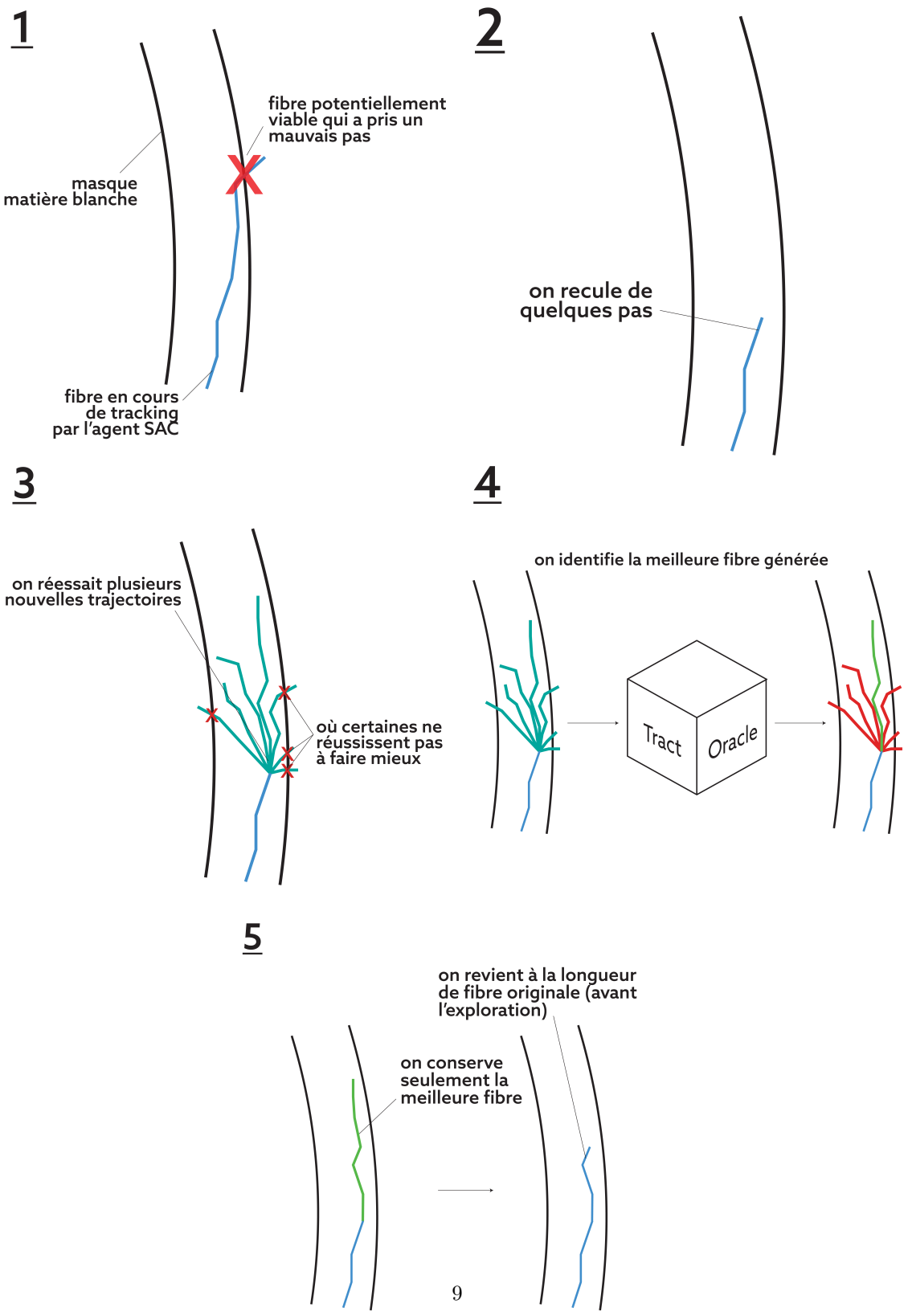


Figure 3: Illustration de l'algorithme de *backtracking* qui permet de corriger la trajectoire de fibres qui s'arrêtent de façon prématurée.

des fibres qui s'arrête prématurément.

Spécification des paramètres

Dans TrackToLearn, le script principal pour valider la performance d'un agent est le script *ttl_validation.py* et la validation permet justement de créer des tractogrammes qui seront pas la suite quantifiés en statistiques. Les paramètres ajoutés par l'introduction de cet algorithme dans l'environnement de *tracking* sont disponibles et peuvent être modifiés directement en faisant appel au script.

- "- -do_rollout" permet d'activer l'algorithme ci-dessus pour corriger la trajectoire de fibres de matière blanche qui s'arrête.
- "- -n_rollouts" permet de spécifier la valeur du nombre de rollouts effectués (NR).
- "- -backup_size" spécifie la valeur de *backup size* (BS).
- "- - extra_n_steps" spécifie la valeur de *forward size* (FS).

5 Résultats

Comme nous avons introduit 3 nouveaux paramètres dans l'environnement de *tracking* de TrackToLearn, j'ai procédé en effectuant une recherche par grille afin de déterminer les paramètres qui semblent offrir une amélioration plus significative par rapport à la méthode de référence qui n'utilise aucun rollout. J'ai limité la recherche par grille à des valeurs de NR = {2, 5, 10}, BS = {1, 3, 5} et FS = {5, 10} puisqu'en faisant quelques expérimentations, je voyais que même avec les valeurs maximales que j'ai utilisé pour la recherche, celles-ci n'étaient pas nécessairement les plus performantes. Le temps pour la recherche par grille était donc raisonnable, avec un temps de quelques heures seulement.

Les 8 combinaisons de paramètres qui résultaient en les plus grandes marges d'amélioration sont affichés dans le tableau 1. Si on compare le nombre de streamlines, on peut voir qu'il semble y avoir un nombre total de streamline légèrement réduits par rapport à la référence. Ce qui est intéressant de constater par contre, c'est que même avec un nombre légèrement plus faible de streamlines, l'utilisation de l'algorithme pour toute combinaisons de paramètres offrait un plus grand nombre de fibres qui étaient valides au final. En comparant avec le ratio initial, le ratio des fibres valides augmente jusqu'à 6,1320 % pour la meilleure combinaison de paramètre trouvée par mon expérimentation. Même si les résultats de toutes les combinaisons de paramètres n'est pas affiché, toutes les combinaisons viennent offrir un avantage sur la méthode originale. Pour le moment, on ne voit pas de claire corrélation ou de tendance claire dans la combinaison de paramètres qui semble donner de meilleurs résultats. A priori, je pensais qu'en ayant une plus grande quantité de rollouts, on trouverait forcément plus de meilleurs trajectoires pour les nouvelles streamlines, mais ça ne semble pas être le cas, puisque les 5 meilleurs méthodes ne sont même pas la valeur maximale du nombre de rollouts testée. Il serait intéressant de tester de manière aléatoire sur une plus grande étendue de paramètres ce qui nous permettrait peut-être d'évaluer un peu mieux des tendances sur certains de ces paramètres.

Si on compare les tractogrammes du meilleur résultat obtenu par rapport à celui de référence dans la figure 4, on obtient quelque chose de relativement très similaire. On peut y voir que les fibres qui semblent moins suivre les gros *bundles* de fibres ne sont pas au même endroits. Globalement

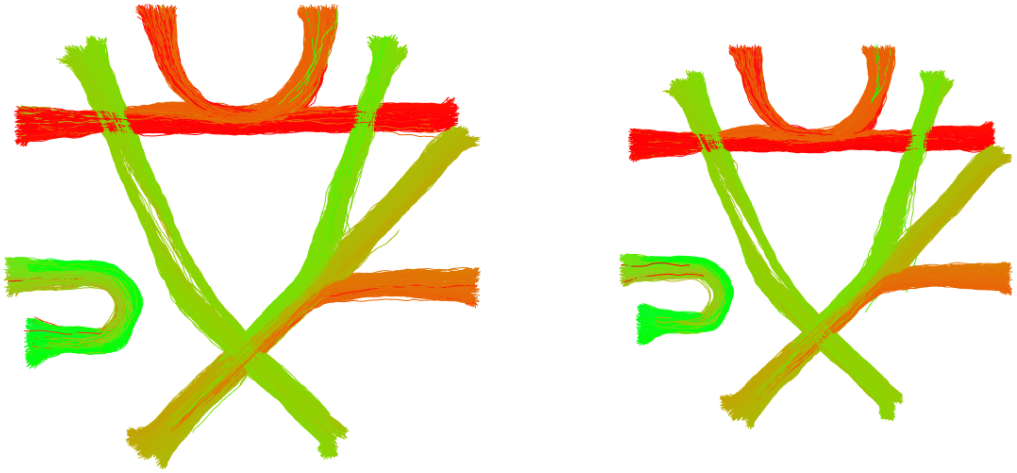
NR	BS	FS	Streamlines	VS	VS_ratio	Amélioration
0	0	0	4067	2797	0.6877	-
2	3	5	3949	2958	0.7491	6.1320 %
5	5	10	3987	2945	0.7387	5.0920 %
2	3	10	3985	2935	0.7365	4.8781 %
5	3	5	3973	2915	0.7337	4.5972 %
2	5	10	3948	2891	0.7323	4.4539 %
10	1	5	3974	2885	0.7260	3.8238 %
2	5	5	3926	2840	0.7234	3.5652 %
5	5	5	3929	2840	0.7228	3.5060 %

Table 1: Meilleures améliorations sur le FiberCup suite à l'usage du *rollout_env*. VS était le nombre total de streamlines qui sont valides, VS_ratio étant le ratio du nombre de streamline valides par rapport au nombre de streamlines totales (colonne Streamlines).

par contre, tout semble très similaire.

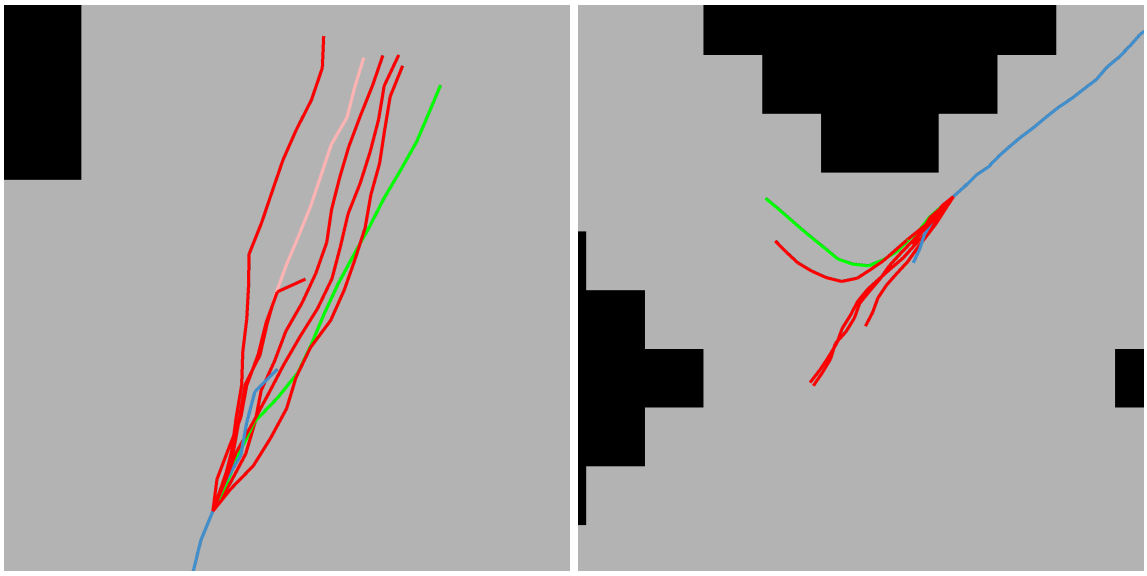
De plus, j'ai aussi effectué quelques rendus dans la figure 5 qui nous permettent de voir certains bouts de l'algorithme en exécution. On y voit en bleu la streamline originale qui s'est arrêtée de façon prématurée dans la matière blanche avec toutes les autres streamlines qui démarrent quelques pas plus tôt dans le traçage de la streamline et qui dévient dans plusieurs sens différents. La streamline qui a été évaluée comme ayant le meilleur score par l'oracle est en vert sur les rendus. Ce qui est particulièrement intéressant avec ces visualisations, c'est qu'elles sont permis de mettre en lumière certains problèmes avec soit l'algorithme ou peut-être même avec l'utilisation de l'oracle dans ce cas. Par exemple, dans le rendu de la figure 5b, la fibre qui est choisie comme étant la meilleure est la fibre qui fait un U en remontant vers le haut. Cependant, en connaissant la structure de FiberCup, il est connu que la fibre de peut pas avoir une telle trajectoire, celle-ci doit normalement continuer dans la même lignée que la trajectoire qu'elle avait quelques pas précédents. Ce problème peut se produire soit parce que l'oracle donne effectivement un score erroné, ou encore l'oracle donne un score équivalent à plusieurs ou toutes les streamlines qui sont en exploration. On vient choisir une streamline en utilisant la fonction *np.argmax* en Python qui vient choisir la première streamline avec un score maximal. Si le score maximal est égal partout, alors la première streamline est choisie par défaut ce qui cause un problème puisque parfois la première streamline ne semble pas du tout idéale dans le contexte. On a même observé des scénarios ou la streamline choisie était celle qui s'arrêtait le plus prématurément parmi toutes ces consoeurs. La résolution de ce problème permettrait de probablement augmenter le taux d'amélioration du nombre de streamlines valides.

De plus, certains tractogrammes produits parmi ma recherche de paramètres semblaient contenir quelques artéfacts de mon implémentation de l'algorithme de d'exploration. Dans la figure **??**, on peut voir que quelques fibres parmi celles générées vont directement au coin bas-gauche de l'image. Ce coin correspond aux coordonnées (0, 0, 0) dans l'espace. Donc, probablement que l'implémentation ne coupe pas assez bien ou qu'il y a une erreur dans la propagation des fibres dans certains cas précis. Ça arrive seulement pour une faible proportion de fibres et seulement dans certains tractogrammes. Donc, probablement que corriger ce petit artéfact permettrait d'améliorer les résultats encore plus étant donnés que ce ne sont pas des streamlines valides. Le problème n'a pas été corrigé par simple contrainte de temps, mais il n'invalide pas les améliorations proposées.



(a) Référence (b) Meilleur résultat du rollout

Figure 4: Comparaison du tractogramme de référence par rapport au meilleur résultat obtenu.



(a) L'oracle semble choisir une bonne streamline. (b) L'oracle choisi une streamline invalide.

Figure 5: Comparaison du tractogramme de référence par rapport au meilleur résultat obtenu.

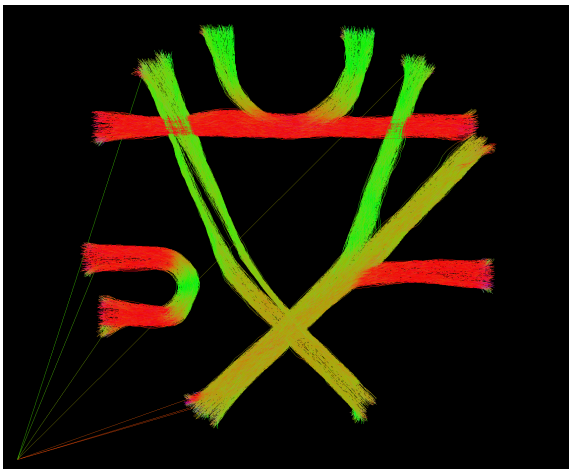


Figure 6: Tractogramme avec des fibres qui vont à la coordonnée (0, 0, 0) dans l'image ce qui résulte d'un problème dans l'implémentation de l'algorithme.

6 Conclusion

Bien que les résultats ci-haut ne sont issus que d'une preuve de concept, ils sont tout de même très encourageant. Ces résultats sont obtenus dans la structure du Fibercup qui est une structure relativement simpliste par rapport à un cerveau au complet. Il y a donc un nombre beaucoup plus grand de fibres qui sont nécessaire pour reconstruire fidèlement un réel cerveau. Si la tendance se maintient aussi pour un environnement de *tracking* d'un cerveau, on pourrait obtenir un avantage considérable sur le taux de fibres qui sont valides au final. Comme c'est un simple ajout, il serait probablement possible d'utiliser cette méthode un peu partout par la suite sans trop se poser de questions puisqu'elle nécessite si peu d'efforts d'implémentation pour être utilisée. Il y a aussi plusieurs scénarios où des centaines de fibres s'arrêtent directement au départ de la propagation des fibres. Pour le moment, les points d'arrêt prématurés très rapprochés des positions initiales des fibres ne sont pas considérés pour éviter de *backtrack* trop loin. Il serait peut-être pertinent d'investiguer ce qui est possible de faire pour les arrêts dès le départ du *tracking* ce qui serait intéressant pour guider un plus grand nombre de fibres dans la bonne direction. Finalement, il serait aussi intéressant d'explorer si on pourrait améliorer les méthodes d'apprentissage par renforcement en utilisant des méthodes *model-based* qui permettent d'approximer directement la distribution de probabilité des transitions dans l'environnement du cerveau. Au final, comme nos algorithmes sont dépendants des modèles qui sont utilisés pour modéliser la diffusion de l'eau dans le cerveau, peut-être pourrait-on bénificier de la flexibilité et la capacité des réseaux de neurones pour approximer plus fidèlement la diffusion de l'eau dans le cerveau?

References

[1] Philippe Poulin et al. "Tractography and machine learning: Current state and open challenges". In: *Magnetic resonance imaging* 64 (2019), pp. 37–48.

[2] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[3] Christopher Berner et al. "Dota 2 with large scale deep reinforcement learning". In: *arXiv preprint arXiv:1912.06680* (2019).

[4] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354.

[5] Long Ouyang et al. "Training language models to follow instructions with human feedback". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 27730–27744.

[6] Antoine Théberge et al. "Track-to-Learn: A general framework for tractography with deep reinforcement learning". In: *Medical Image Analysis* 72 (2021), p. 102093. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2021.102093. URL: https://www.sciencedirect.com/science/article/pii/S1361841521001390.

[7] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[8] Jon Haitz Legarreta et al. "Filtering in tractography using autoencoders (FINTA)". In: *Medical Image Analysis* 72 (2021), p. 102126.

[9] Laurent Petit et al. "The structural connectivity of the human angular gyrus as revealed by microdissection and diffusion tractography". In: *Brain Structure and Function* 228.1 (2023), pp. 103–120.

[10] Antoine Théberge, Maxime Descoteaux, and Pierre-Marc Jodoin. "TractOracle: anatomically-plausible reward function for reinforcement learning-based tractography." Work in progress. 2023.

[11] Gabriel Girard et al. "Towards quantitative connectivity analysis: reducing tractography biases". In: *Neuroimage* 98 (2014), pp. 266–278.