Rapport du projet
Détection de pattern d'épistasie

Louison Fresnais - François Courtin

M2 Bioinformatique, Université de Nantes

Février 2019

1 Introduction

Au cours de ce projet, nous avons implémenté deux méthodes de recherche de pattern d'épistasie :

— Une méthode de recherche de pattern d'épistasie nommée SMMB-ACO qui combine à la fois une recherche stochastique similaire aux méthodes de random forests et une méthode bayésienne de couverture de markov.

— Une méthode de recherche de pattern d'épistasie par un algorithme génétique qui correspond à une méthode dite évolutionnaire. Celle-ci utilisera les solutions présentes dans sa population pour effectuer des crossing-over et parfois des mutations et ainsi générer de nouvelles solutions potentiellement meilleures que les précédentes.

Dans ce rapport, nous allons développer les particularités de chacune des implémentations des méthodes que nous avons réalisées ainsi que les améliorations que nous proposons.

2 Simulation de données naïves

L'objectif de ce script est de produire un jeu de données avec un nombre de SNPs influençant le phénotype donné par l'utilisateur. Ce script vient avec un parseur d'arguments. Celui-ci permet entre autre de fournir une présentation du script et des arguments à passer en utilisant l'argument -h ou –help. Si aucun argument n'est entré, le programme demandera à l'utilisateur d'entrer, en les décrivant, les arguments manquants jusqu'à ce qu'ils soient tous fournis.

L'élément principal de la génération de données naïve est de pouvoir générer des données de génotype avec un nombre donné de SNPs dont la combinaison de valeurs va avoir un impact important sur le phénotype de l'individu. Afin de réaliser ceci, nous avons implémenté une génération de coefficients de régression logistique aléatoire (qui prend en compte une taille de pattern d'épistasie donnée par l'utilisateur). Nous allons donc chercher des modèles qui permettent de discriminer correctement les individus malades et non malades selon des combinaisons de 0,1 ou 2.

Un des modèles sélectionné par la fonction fit_relevant_logit est ensuite utilisé pour générer les données. On s'assure de générer des nombres de cas et de contrôles égaux et dont la somme correspond au nombre d'individus demandé par l'utilisateur.
Différentes techniques sont utilisées afin d'anticiper d'éventuelles erreurs de l'utilisateur telles qu'un oubli de paramètre ou des noms de fichiers identiques.

Cependant, notre méthode de génération de données naïves pourrait être améliorée en suivant les consignes qui ont été données en toute fin de projet et qui consistaient à générer des données naïves avec des SNPs causaux avec des Betas fixés.

3 Algorithme Génétique

Particularités de notre implémentation

Afin d'optimiser l'efficacité de notre algorithme génétique, nous avons choisi d'initialiser la population de base avec des solutions uniques. Nous avons fait de même pour la sélection des parents parmi la population à chaque itération. Ces deux particularités ont posé problème lors de l'exécution de la méthode sur des données ne présentant que peu de SNPs (moins de 30). En effet, avec un pattern

de taille 2 et seulement 28 SNPs, le nombre de solutions uniques n'est que de 378. Pour parer à ce problème nous avons fixé la taille de la population au maximum (calculé grâce à une combinatoire) lorsque celle rentrée par l'utilisateur était trop grande.

Pour ce qui est de la sélection de parents uniques, lorsque le nombre de parents, rentré par l'utilisateur, sélectionnés à chaque itération était trop important par rapport à la taille de la population, il devenait impossible de sélectionner des parents tous différents. Le problème vient du fait qu'au fur et à mesure de l'exécution de la méthode, la population se remplit de solutions identiques. Pour régler le problème, nous avons indiqué à l'algorithme de ne tenter de trouver un parent différent que vingt fois par sélection de parent. Au delà, il est autorisé à sélectionner des parents identiques.

Une fois de plus pour améliorer l'efficacité de l'algorithme et la qualité des sorties de la méthode, nous avons ajouté un paramètre permettant d'indiquer à l'algorithme la probabilité de choisir une "mauvaise solution" comme parent. Ainsi, si cette probabilité est fixée à 0.05 par exemple, l'algorithme produira de nouvelles solutions à partir de solutions déjà considérées comme "bonnes" dans 95% des cas. Le seuil permettant de définir une solution comme étant bonne est fixé à partir de la médiane des scores g2 des solutions composant la population. Cette médiane est recalculée à chaque itération pour augmenter la qualité moyenne des solutions considérées comme bonnes.

Améliorations à apporter

Pour optimiser notre algorithme, il serait utile d'implémenter une clause de sortie de la méthode avant la fin des itérations définies par l'utilisateur dans le cas où l'on aurait un certain nombre de solutions ayant une qualité supérieure à un seuil fixé. Cela permettrait de voir le temps d'exécution se réduire. Il n'est malheureusement pas possible d'effectuer cette modification pour le moment à cause d'un problème de p-valeurs que nous détaillerons par la suite.

La solution trouvée pour palier au problème de sélection de parents uniques en cas de population de petite taille fonctionne mais n'est pas optimale. Une implémentation permettant d'indiquer une liste de parents encore disponibles à la fonction les sélectionnant permettrait de voir directement si des parents n'ont pas encore été sélectionnés. Ainsi, si la liste se vide, nous pourrions ou bien sélectionner plusieurs parents identiques, ou bien réduire le nombre de parents uniques sélectionnés au maximum possible. Cela éviterait une perte de temps due aux vingts essais dont la fonction dispose, mais cela permettrait aussi, dans le cas où l'on réduit le nombre de parents, d'améliorer la qualité des enfants créés (car produits uniquement à partir de parents différents).

Afin d'obtenir un algorithme relativement rapide et non excessivement gourmand en mémoire, nous avons décidé de l'implémenter de façon à ce qu'il fonctionne uniquement avec un pattern d'épistasie de taille 2 ou bien de taille 3. Ainsi, la taille des solutions dans la population de varie pas. Cela permet d'avoir des tables de contingences relativement petites et donc rapides à calculer et peu encombrantes. Cependant, cela limite la création de nouvelles solutions par crossing-over. En effet, si nous avions autorisé des tailles de patterns variables, l'algorithme aurait pu créer des solutions à partir de plus de SNPs et contenant plus de SNPs, maximisant ainsi les chances d'obtenir les SNPs causaux. Nous aurions pu implémenter une fonctions gérant une taille maximale à ne pas dépasser lors de la création de nouvelles solutions pour faire un compromis entre qualité des résultats et optimisation matérielle de la méthode.

Enfin, nous avons décidé de classer les solutions finales par p-valeur pour nous accorder aux autres projets réalisés par nos collègues et aux consignes données, cependant, dans le cas de p-valeurs égales à 0, il serait préférable de les classer par score. En effet, le score peut varier en fonction de la taille du pattern mais comme dit précédemment, nos patterns sont de tailles constantes. Classer les solutions dont la p-valeur est de 0 par score ne poserait donc pas de problèmes dans notre cas.

3.1 Problèmes

Un problème que nous avons encore aujourd'hui réside dans les p-valeurs. En effet, lorsque l'un des SNPs causaux est présent dans une solution, la plus part du temps, le score g2 affecté à cette solution est très élevé. Cela entraîne donc une p-valeur très faible mais aussi nulle dans la plus part des cas. Or une p-valeur nulle perd tout son sens statistiquement. Dans certains cas nous avons même des score g2 ayant une valeur de plus l'infini. Nous nous retrouvons donc obligés de faire un choix, ou bien affecter une valeur très grande au score g2 et donc une p-valeur de 0, ou affecter un score de 1 et donc une p-valeur de 1, ce qui entraînera l'élimination de la solution. Nous avons choisi la deuxième option par soucis de significativité statistique crédible. Cependant, nous n'avons pas sorti toutes les solutions ayant des scores élevés et des p-valeurs à 0 car la majorité des patterns considérés comme true positif sont dans ce cas là.

Enfin, la méthode parvient à trouver assez souvent les patterns causaux lorsqu'ils sont d'une taille 2 SNPs mais lorsque cette taille atteint les 3 SNPs, les patterns causaux se font plus rares dans les résultats. Pour remédier à ce problème il faudrait augmenter les paramètres rentrés dans le fichiers de paramètres (plus d'itérations, plus de parents,...). Ce problème est encore plus flagrant avec les données gamètes où elle ne trouve quasiment jamais les patterns de taille 3.

4 SMMB-ACO

Brève description de la méthode

L'algorithme SMMB-ACO est un algorithme de détection de pattern d'épistasie dont l'objectif est d'apprendre un nombre important de couvertures de Markov sous-optimales dans le but d'en ressortir une optimale en fin d'algorithme. Il s'agit donc d'une méthode non déterministe. Dans notre méthode, toutes les couvertures de Markov sont stockées mais triées selon leur p-value.

Particularités de notre implémentation

L'une des particularité majeure de notre version de SMMB-ACO réside dans la méthode de récolte des résultats. En effet, l'implémentation réalisée par Clément Niel fait appel à une méthode complexe qui permet d'évaluer la qualité de chaque markov blanket produite. Dans notre cas, nous avons choisis d'évaluer les résultats en fin de phase backward puisque sont ajoutées uniquement les markov blanket dont les p-valeurs sont inférieures à 0.05 et qui ne sont pas déjà présentes dans la map de résultat (si présente, on ajoute +1 au nombre d'occurrences). De cette manière, nous espérons gagner du temps de calcul tout en conservant des résultats corrects puisque ajoutés en fin de méthode et sans doublon.

Nous avons également fait le choix d'utiliser au maximum les propriétés d'accès direct des vecteurs afin d'avoir un algorithme le plus rapide possible. Cela a d'ailleurs pu poser quelques difficultés lors de l'adaptation du module de statistiques que vous nous avez fourni puisqu'il utilisait majoritairement des listes. (code de Clément Niel).

Enfin, en ce qui concerne la génération des combinaisons de pattern d'épistasie, nous avons choisi de réaliser toutes les combinaisons à partir de la taille du vecteur de snps pour une itération ACO donnée et de la taille du pattern d'épistasie. Cela nous donne des combinaisons d'indices auxquelles nous pouvons ensuite accéder rapidement en accès direct.

Améliorations à apporter

Nous avons rencontré de nombreuses difficultés dans la compréhension de la méthode SMMB-ACO en début de projet ce qui a limité notre marge de manoeuvre en terme d'optimisation de notre code. C'est pourquoi, il y a probablement des éléments que nous pourrions optimiser afin de garantir des temps de calculs corrects avec des jeux de données contenant des milliers de SNPs.

Il serait également intéressant de revenir sur les méthodes statistiques employées dans cet algorithme. En effet, notre méthode de validation de la qualité d'une markov blanket pourrait être de meilleure qualité en testant la qualité de chaque solution par rapport aux autres en fin de programme.

Comme nous avons pu le constater en lançant l'algorithme sur tous les jeux de données et sur 100 fichiers sur le serveur bioinfo, notre méthode génère de très nombreux fichiers (de faible taille) mais qui risque de rapidement atteindre le nombre de fichier maximum fixé par l'administrateur. Nous pourrions alors envisager une compression (en multithread avec pigz) du dossier contenant les résultats à la fin du script python qui permet le lancement du programme complet (100iterations SMMB-ACO + évaluation) et une suppression du dossier. Cela augmenterait le temps total mais permettrait d'éviter un problème certain lors du passage à l'échelle avec de très gros jeux de données.

Résultats

Pour des raisons de temps de calcul, nous avons fait tourner SMMB-ACO sur tous les jeux de données avec 50 itérations et systématiquement les mêmes paramètres. Les résultats sont plutôt bons pour les jeux de données naïves avec 2snp causaux mais ils se dégradent rapidement avec les jeux de données gamètes avec une MAF faible ou avec 3snp causaux. Il faudrait adapter les paramètres qui ont étés optimisés au départ pour des jeux de données de 2snp causaux en simulation naïve. En ce qui concerne les temps, ils ne varient pas selon le nombre d'itérations sur un fichier. Pour des jeux de données 2snp causaux, une itération sur un fichier dure environ 1 seconde. Pour 3snp causaux, on est à environ 3.5 secondes.

Problèmes

Il ne reste qu'un seul problème que nous avons identifié mais pas encore réussi à résoudre. Il s'agit d'une fuite mémoire qui survient à cause du calcul parallèle réalisé par openmp. Après recherche des fuites mémoires à l'aide de l'outil Valgrind, nous avons pu identifier qu'en ajoutant un pragma omp critical (qui permet de faire réaliser un bloc de code uniquement par un thread) dans la méthode intitulée : select_snp_in_distrib_prob(float prob) le programme ne pouvait plus se compiler(variable non déclarée dans cette vue, alors qu'elle est déclarée). En retirant le calcul parallèle, cette fuite mémoire disparaît. Cependant, étant donné que notre algorithme est suffisamment performant pour détecter correctement les patterns d'épistasie avec des paramètres modestes et en utilisant peu de mémoire (168MB en permanence sur le serveur bioinfo avec 50 threads), nous avons pu réaliser du calcul parallèle malgré ce problème. Enfin, il semble que le serveur bioinfo n'ai pas permis de gagner du temps de calcul à cause de son utilisation par plusieurs personnes. Les calculs ont étés plus rapides sur nos machines personnelles.

5 Évaluation des méthodes

L'objectif de ce script est d'évaluer la qualité de notre méthode à déceler les pattern de SNPs causaux à partir des données de simulation naïve. Cette qualité sera évaluée à l'aide du calcul d'une f-measure et d'un power. Ces deux calculs dépendent du nombre de True Positif, de False Negatif et de False Positif décelés pour chaque fichier.

Nous avons décidé de créer un script en python par méthode permettant de lancer l'algorithme sur un jeu de données entré en paramètre qui appellera ensuite un script d'évaluation des résultats commun aux deux méthodes. Ainsi avec une seule commande, nous sommes en mesure de lancer la méthode de notre choix sur tous les fichiers présents dans un jeu de données, le nombre de fois voulu et d'obtenir en sortie les résultats de la méthode pour chaque exécution sur chaque fichier ainsi que l'évaluation de cette méthode pour chaque fichier.

Les deux méthodes trouvent régulièrement le pattern causal si ce n'est avec les jeux de données "Gamètes" de patterns causaux de taille 3.

6 Conclusion

En conclusion, nous avons réussi à implémenter les deux méthodes et à obtenir des performances de détection de pattern d'épistasie et de temps de calcul correctes. Ce projet nous a permis d'apprendre le C++ et nous avons pu constater nos progrès tout au long du projet. Ainsi, si l'on devait continuer le projet, nous pourrions encore mieux l'optimiser à l'aide des compétences acquises jusqu'ici.

Avec plus de temps, il nous paraîtrait intéressant de comparer nos deux méthodes entres elles et ainsi observer si l'une sacrifie la vitesse d'exécution pour la qualité et inversement, et donc pouvoir s'inspirer des mécanismes de l'une pour améliorer l'autre.