Outil de formulation Wuji & Co

Bruno Coupier

8 décembre 2019

Table des matières

Chapitre 1

Bases de données

Les approches algorithmiques envisagées au Chapitre 2 requièrent la constitution de trois bases de données :
• La base de données "Profils" : constituée par les profils nutritionnels génériques définis par les nutritionnistes.
• La base de données "Ingrédients", constituées initialement par la base de données nutritionnelles française de reference CIQUAL [1].
• La base de données "Recettes", constituées initialement en téléchargeant des recettes de l'API d'Edamam (www.edamam.com), puis en scrapant des sites web de recettes (ex :www.allrecipes.com).

1.1 Base de données Profils

1.1.1 Constitution

Cette base a été définie par les nutritionnistes. Elle est constituée de 58 profils nutritionnels.

1.2 Base de données Ingrédients

1.2.1 Constitution

Cette base est constituée initialement par la base de données CIQUAL, de laquelle les ingrédients transformés et les plats préparés ont été retirés pour ne garder que les ingrédients élémentaires.

1. https ://ciqual.anses.fr/

1.2.2 Traduction

Il est nécessaire de traduire les ingrédients en anglais ainsi que de définir des mots-clefs en anglais utilisés pour la recherche de recettes par requêtes sur l'API d'Edamam, ainsi que pour effectuer l'analyse des lignes d'ingrédients des recettes et les faire correspondre aux ingrédients de référence de notre base de données. La première traduction a été obtenue automatiquement en utilisant le module python googletrans, puis la traduction a été vérifiée et corrigée, notamment celle des mots-clefs a été revue et repensée pour déterminer les mots-clefs les plus concis et spécifiques possible (voir 1.4) ...En fait je viens de découvrir qu'il y avait une version anglaise disponible pour la table CIQUAL ... On utilisera la traduction officielle de préférence donc ...

1.2.3 Nettoyage

1.2.4 Enrichissement

- Ajout d'ingrédients nouveaux.
- Ajout d'informations supplémentaires sur les ingrédients.

1.3 Base de données Recettes

1.3.1 Constitution

Nous avons identifié la base de données d'Edamam comme étant la plus adéquate pour la constitution d'une base de données de recettes initiales. La raison en est qu'Edamam fournit une première analyse nutritionnelle complète des recettes ainsi qu'une normalisation des quantités des ingrédients en grammes. Nous avons utilisé l'API d'Edamam pour effectuer des requêtes de recettes sur une sélection d'ingrédients de la base Ingrédients (la colonne "search keyword2" de la bdd Ingrédients contient les keywords de recherche par ingredient sur l'API d'Edamam), notre objectif étant de constituer une collection de recettes la plus variée et la plus riche possible pour être utilisée comme source d'information pour notre évaluation des bonnes associations d'ingrédients du point de vue du goût. Nous nous sommes également efforcés de collecter les recettes en tenant compte de la cuisine d'origine des recettes. En effet, les associations d'aliments sont caractéristiques de la cuisine d'origine des recettes et il est donc important d'en tenir compte

pour évaluer la satisfaction des critères de goût d'une cuisine spécifique. Des champs de recherche par type de cuisine (pays d'origine), type de repas ("mealType" :Breakfast, Lunch, Dinner...) et types de plats ("dishtype" :Cereals, Salad, Soup...) sont disponibles sur l'API Recipes d'Edamam [2]. Le programme utilisé pour le téléchargement des recettes de la base d'Edamam est `request_multiple_Edamam.py`. Les fichiers individuels téléchargés sont alors concaténés en un seul dataframe par le programme `data_to_df.py`.

1.4 Normalisation des lignes d'ingrédients des recettes

Cette section concerne les méthodes pour établir la correspondance entre les lignes d'ingrédients de recettes récupérées sur le web et les ingrédients de référence de la base de données Ingrédient (initialement constituée par les aliments de la base CIQUAL). C'est une étape nécessaire à l'identification des ingrédients qui composent les recettes collectées sur le web dans le but de leur utilisation pour déterminer les meilleures association d'aliments ainsi que pour évaluer leurs apports nutritionnels à partir de notre base de référence.

1.4.1 Méthodes

Utilisation du programme ingredients.py

Dans la thèse de Sarnhil, ce programme est utilisé dans un but de normalisation des ingrédients c'est à dire pour extraire des lignes d'ingrédients de recettes non structurées, les noms, quantités et autres informations associées aux ingrédients. La décomposition ne fonctionnant pas toujours idéalement, les noms des ingrédients ne sont pas systématiquement bien identifiés. Par contre, nous pourrions utiliser ce programme pour la détermination de la quantité qui fonctionne bien généralement.

Utilisation du programme ingredient-phrase-tagger du New York Times

Le New York Times a utilisé une méthode basée sur l'algorithme Conditional Random Field (CRF) pour faire l'analyse des lignes de recettes, et identifier les noms des ingrédients, les quantités et unités, ainsi que des informations annexes. Même remarque que pour la méthode précédente concernant l'identification du nom de l'ingrédient.

2. https ://developer.edamam.com/edamam-docs-recipe-api

Méthode directe par reconnaissance de mots-clefs avec python re module

Contrairement aux méthodes précédentes qui visent à déterminer les informations relatives aux ingrédients de la recette en se basant sur la structure de la ligne d'ingrédient, la méthode directe consiste à reconnaître directement des mots-clefs associés aux ingrédients de référence de la base Ingrédient en utilisant le module python re (regex).
Le programme ingredientMatcher.py permet d'effectuer la correspondance entre une ligne d'ingrédient d'une recette et un ou des ingrédients de la base Ingrédients de référence. En général, une ligne d'ingrédient contient un seul ingrédient, mais il se peut que plusieurs ingrédients alternatifs soient suggérés ou bien que plusieurs ingrédients soient groupés dans une seule ligne (ex : "herbs (marjoran, parsley [...])", ou bien "salt and pepper"). Tous les ingrédients seront retenus dans le cas de reconnaissances multiples. La quantité sera alors partagée entre les matchs obtenus.
La procédure générale de matching peut se décrire ainsi : on passe en revue tous les mots clefs de la base Ingrédient et liste les matchs obtenus avec une ligne d'ingrédient de la recette, puis, dans un deuxième temps, si il y a plusieurs matchs, on utilise les informations complémentaires de la colonne modifiers pour sélectionner un unique ingrédient parmi les ingrédients qui ont matchés.
Dans le cas où l'on obtient des matchs avec des lignes de la base Ingrédients qui ont le même mot clef, on fait l'inventaire pour chaque ligne des termes de la colonne modifiers qui se retrouve dans la ligne d'ingrédient de la recette. La ligne de la base Ingrédients qui comptabilisera le plus de matchs sera retenue. En cas d'égalité, la ligne avec le plus petit index est retenue.
Si les éléments modificateurs (colonne modifiers) n'ont pas permis de trouver une correspondance plus spécifique, c'est à dire dans le cas où aucun matchs n'a été décelé entre les modificateurs et la ligne d'ingrédient de la recette, nous sélectionnerons en priorité la ligne qui correspond à l'ingrédient identifié comme générique et tagué avec le modificateur "generic" dans la colonne modifiers (ex : pour le mot "farine" dans une ligne de recette, le match est obtenu avec le mot clef "flour|wheat flour|soft wheat flour|bread wheat flour" qui a 6 correspondances dans la base Ingrédient. Cependant, seul l'ingrédient, dont les modificateurs sont "all-purpose,generic", sera retenu).
Pour un certains nombre d'ingrédients pour lesquels il n'y a pas d'ambiguité et qui sont soit à l'état cuit ou cru dans la base Ingrédient de référence ("cooked" or "raw"), il n'est pas nécessaire de recourir au tag "generic" et la sélection se fait en gardant en priorité la ligne correspondant à l'ingrédient cuit (mots clefs "cooked","mashed","fried","boiled","roasted","baked"), ou

l'ingrédient cru en second choix (mots clefs "raw","fresh"). La première ligne (index le plus petit) est retenue dans le cas de plusieurs lignes avec le modificateurs cuit, dans le cas de plusieurs lignes avec le modificateurs crus si aucun ingrédient cuit n'a été trouvé.
Nous voulons attirer l'attention sur le fait que la colonne keyword et la colonne modifiers sont des colonnes d'expressions regex. En particulier, outre les caractères spéciaux utilisés, les espaces sont importants pour éviter des matchs erronés (par ex. pour éviter que "eel" ou "ling" matchent avec des termes comme "peeled" ou "drizzling", les keywords correspondant incluent des espaces (" eel "," ling ").
Un bon choix de mot clef (colonne keywords dans la base Ingrédients) est important pour garder le maximum de spécificité sans risquer de rater des matchs. Si le mot clef est trop spécifique, on risque de rater un match... si le mot clef est trop général on risque d'avoir trop de matchs (ex : le mot-clef "rice" est trop général alors que le mot clef "brown rice" est plus spécifique et convenable puisqu'il sera généralement spécifier dans les recettes utilisant du riz complet). On a souvent le choix entre choisir un mot-clef plus spécifique dans la colonne keyword, ou choisir un mot-clef moins spécifique et ajouter des modificateurs dans la colonne modifiers. Cependant, un mot plus spécifique du départ réduira le nombre de matchs initiaux et évitera une erreur de correspondance. Dans la mesure du possible, il vaut donc mieux choisir un mot-clef le plus spécifique possible, quitte à utiliser des synonymes ou variations d'appellation dans la colonne keyword (ex : pour le riz complet le mot-clef dans la colonne keyword est "brown rice|wholegrain rice".
On doit aussi vérifier que les mots-clefs des différents ingrédients ne créent pas de conflits entre eux. Par exemple, pour le mot clef "pepper" qui peut signifier le poivre et le poivron en anglais, on a choisi de spécifier "bell pepper" pour le poivron pour éviter un conflit entre les deux ingrédients. Ainsi, avant de modifier des mots clefs existants ou d'ajouter des nouveaux mots clefs, il faut prendre en compte les interactions possibles avec les autres mots clefs.
Tous les ingrédients rencontrées dans les recettes ne sont pas recensés dans la base ou bien il peut y avoir eu des erreurs de traductions ou des choix de mots clefs trop spécifiques. Les lignes d'ingrédients qui n'ont pas trouvées de correspondance sont sauvegardées dans un fichier ingrNoMatch.csv pour un examen postérieur et une amélioration de la base Ingrédient en vue d'une identification plus complète.
L'évaluation de la performance du matching est obtenue en comptabilisant les analyses réussies des lignes d'ingrédients d'une recette par la procédure décrite auparavant. L'obtention d'une résolution ne garantit pas la justesse du résultat (possibilité de tester avec résultats obtenus par Edamam ?). Nous

devons nous efforcer de progressivement améliorer la base d'ingrédients (modification mots clefs et ajout d'ingrédients) pour améliorer la justesse des matches. On associe les codes suivants aux résultats d'analyses :

- code "0" pour "single match found".
- code "1" pour "several generic error".
- code "2" pour "keeping generic match".
- code "3" pour "keeping first raw or fresh".
- code "4" pour "modifier not helping to decide the exact line".
- code "5" pour "using modifiers, single keyword match found".
- code "6" pour "all matching indices removed error".
- code "7" pour "No match found in the Ingredient Base CIQUAL".
- code "8" pour "keeping first cooked".

Seules les résolutions de code "0","2","3","5","8" sont considérées comme des matchs réussis, les autres ne sont pas comptabilisées.

['1 cup uncooked long-grain rice',	[[(9105, 'rice|white rice', 185.0, 5)],
'1 cup blanched almonds',	[(15000, 'almond', 145.0, 3)],
'2/3 cup white sugar, plus more to taste',	[(31016, 'sugar|white sugar', 133.3, 0)],
'1 1/2 teaspoons vanilla extract',	[(11098, 'vanilla', 6.3, 2)],
'1 teaspoon ground cinnamon',	[(11025, 'cinnamon', 2.6, 0)],
'1 quart water',	[(18066, 'water', 946.3, 0)],
'2 cups milk']	[(19024, 'milk|whole milk', 488.0, 2)]]

TABLE 1.1 – Lignes d'ingrédients|Matchs trouvés par IngredientMatcher.py

Sur l'exemple de la table 1.4.1, on a matché 100% des lignes d'ingrédients et un seul match par ligne a été retenu. Pour chaque match, le résultat est un tuplet : le premier chiffre indique le code alim de la base CIQUAL qui identifie l'ingrédient de façon unique. Puis, vient le mot clef qui a permis le match et indique de quel ingrédient il s'agit. Ensuite, la quantité en gramme est indiquée, puis le code de résolution qui indique la manière dont le match a été résolu (codes indiqués plus haut).

Tests de la méthode directe

Postérieurement à nos premières analyses des correspondances entre lignes d'ingrédients des recettes et les ingrédients de la base de référence (CIQUAL), Edamam.com a ajouté au fichier JSON des recettes un champ label identifiant l'ingrédient de la ligne de recette. En utilisant ces identifiants (labels) à la place des lignes d'ingrédient complètes, nous réduisons le nombre de matchs par ligne d'ingrédients. Plusieurs tests de l'analyse des matchs ont été effectués : 1.1 compare le nombre de matchs par lignes d'ingrédients obtenus en utilisant la ligne complète de la recette ou bien en utilisant les labels d'Edamam. 1.4.1 fait l'inventaire des types de résolutions obtenues par IngredientMatcher.py en utilisant les lignes d'ingrédients complètes ou les labels. Si l'on compare les matchs communs entre l'analyse faite à partir des labels et l'analyse faite à partir des lignes d'ingrédients entières, on trouve un match commun sur environ 75% des lignes d'ingrédients c'est a dire que les deux méthodes donnent des résultats en accord dans 75% des cas environ.

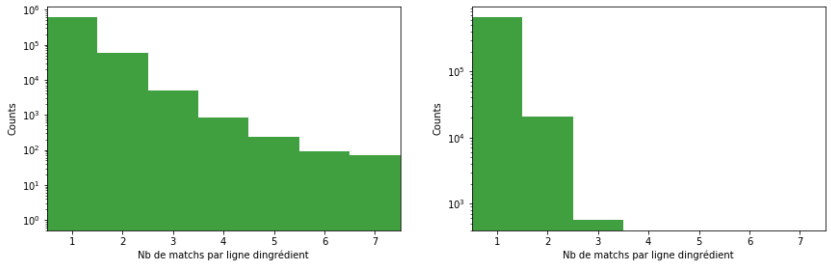

FIGURE 1.1 – Analyse à partir des lignes d'ingrédients des recettes (gauche), analyse à partir des labels d'Edamam (droite).

Type de résolutions	Nb (matchs par ligne)	Nb (matchs par label)
0	235778	256739
2	152577	171082
3	197792	196865
4	6575	5921
5	47470	73132
6	9	4
7	68424	56446

TABLE 1.2 – Types de résolutions trouvées par IngredientMatcher.py à partir des lignes d'ingrédients entières ou bien des labels provenant de l'analyse d'Edamam.com

Chapitre 2

Approches algorithmiques

L'objectif de ces algorithmes est de générer des formules culinaires (recettes) qui doivent satisfaire au mieux les besoins nutritionnels d'une population de profils nutritionnels, sur un nombre de repas variables. La réalisation de recettes savoureuses est bien évidemment de toute première importance et repose sur la détermination de bonnes associations d'aliments à partir de recettes collectées sur le web.

Nous avons envisagé différentes approches algorithmiques pour atteindre cet objectif. La première approche nous a permis de nous familiariser avec le problème et de tenter une approche "force brute" pour trouver des recettes. Les autres approches sont basées sur des méthodes stochastiques qui évitent l'explosion combinatoire rencontrée dans une approche systématique, et permettent une optimisation au travers d'une exploration aléatoire de l'espace des recettes plus ou moins guidée.

2.1 Algorithme systématique

Avant de se lancer dans l'implémentation d'algorithmes plus complexes (exploration aléatoire, apprentissage renforcé...), nous avons décidé d'implémenter un générateur simple à partir d'un nombre limité d'ingrédients qui prend en compte les règles de formulation imposées par les nutritionnistes. L'idée est d'utiliser certaines des règles de formulation sur la composition des recettes (utiliser deux féculents différents dont un légumineux, une viande ou oeuf ou poisson ...) en partant d'un nombre limité d'ingrédients pour générer systématiquement des recettes, puis de sélectionner certaines des recettes dont les ingrédients s'apparient bien entre eux selon une mesure basée sur les corrélations entre ingrédients dans la base de données recettes (Edamam). La dernière étape consiste à ne retenir que les recettes qui satisfont les critères

définis par les nutritionnistes (indicateur PRAL <0, SAIN>5, 70% des kCal d'origine végétal, ... et satisfaction des besoins correspondant aux profils nutritionnels des catégories de population envisagées). Le schéma 2.1 présente les grandes étapes du fonctionnement du générateur systématique.

2.1.1 Génération systématique

Une sélection de la base de données "recettes" est effectuée au préalable pour retenir les recettes correspondant aux types de plats que l'on veut générer (ex : déjeuners français). La génération systématique se fait initialement en combinant toutes les possibilités de choix d'ingrédients parmi les familles d'ingrédients devant entrer dans la composition des plats selon les nutritionnistes. Ainsi pour un plat principal on génère toutes les combinaisons de deux féculents dont un légumineux, une source de protéine animale (viandes, oeufs, poissons) ou végétale, et un légume. On se limite à une sélection d'ingrédients pour chacune des familles pour réduire l'explosion combinatoire initiale à 1 million voir quelques millions de recettes générées. Pour cela, nous retenons seulement les ingrédients qui sont rencontrés un certains nombres de fois parmi les recettes de la base de données. De nouveaux ingrédients sont ajoutés une fois que les recettes ont subi une première sélection par des critères d'appariement des ingrédients entre eux comme expliqué dans la section suivante.

2.1.2 Sélection des "meilleures" recettes

La deuxième étape consiste à réduire le nombre de recettes initiales en choisissant les recettes dont les ingrédients s'apparient le mieux du point de vue du goût. Le score d'appariement est déterminé en utilisant les coefficients de corrélation calculés à partir des recettes de la base de données. La sélection des recettes par pays, puis par plat, effectuée auparavant, permet de garder des recettes qui correspondent aux association d'ingrédients typiques que l'on veut générer.

2.1.3 Détermination des quantités d'ingrédients

Edamam fournit des quantités d'ingrédients en gramme déterminées à partir de leur analyse des lignes d'ingrédients des recettes qu'ils ont scrapées depuis des sites en ligne. Pour notre générateur systématique, notre première idée pour définir les quantités d'ingrédients, était d'utiliser les poids moyens déterminés à partir des recettes d'Edamam. Nous avons constatés que la base de données Edamam contient de nombreuses erreurs sur les quantités.

D'autre part, certaines quantités moyennes sont calculées à partir de trop peu de recettes pour obtenir une moyenne statistique fiable. De plus, lors du processus de matching entre les lignes d'ingrédients des recettes et les ingrédients de la base ingrédients de référence (CIQUAL), en cas de matchs multiples pour une ligne d'ingrédient de recette, la quantité associée à cette ligne d'ingrédient est répartie entre tous les matchs (voir 1.4.1). Il en résulte que les quantités moyennes calculées à partir des recettes de la base de donnée sont en général bien inférieures aux quantités recommandées par le GEMRCN (http ://www.gemrcn.fr/). Les figures 2.2, 2.3 et 2.4 montrent des histogrammes comparatifs des poids moyens par ingrédients aux poids recommandés par GEMRCN. Dans une première approche, on constitue des recettes en utilisant des poids moyens, pour les raisons juste invoquées, nous préférerons utiliser les quantités recommandées par GEMRCN plutôt que les poids moyens calculés à partir de la base de données de recettes Edamam.

2.1.4 Sélection des recettes satisfaisant les critères nutritionnels

On effectue le calcul de l'apport nutritionnel des recettes à partir de la valeur nutritionnelle des ingrédients de référence cuits en priorité puis crus en second choix. Il est intéressant d'examiner l'effet de la cuisson sur les grandes familles d'aliments, de façon à estimer l'erreur que l'on est susceptible de faire en se basant sur la valeur nutritionnelle des aliments crus ou cuits. Le boxplot de la figure 2.1.4 montre l'effet moyen de la cuisson sur la valeur nutritionnelle des légumes et celui de la figure 2.6 sur celle des viandes.

2.1.5 Analyse nutritionnelle des recettes de la base de données

On peut se demander combien de recettes de la base de données d'Edamam satisfont les critères des nutritionnistes. En sélectionnant uniquement les repas consistants du type "Main Dish", on obtient le résultat de la figure 2.8. Ce diagramme montre que seul environ 30% des recettes de plats principaux d'Edamam sont susceptibles d'apporter les calories correspondant à 25% de l'apport journalier minimal d'un individu. On s'attendait à ce que la plupart des plats principaux considérés remplissent cette condition. Après examen des quantités moyennes bien en deçà des quantités recommandées par GEMRCN (2.1.3), nous pouvons comprendre que la nutrition ne fait que révéler une mauvaise évaluation des quantités par individu pour ces recettes. Ou bien, comme expliqué à la section 2.1.3, et malgré le fait que les portions

de GEMRCN doivent être revues à la baisse (discussion avec Eric : les portions GEMRCN sont déterminées pour des plats composés principalement d'un ou deux ingrédients principaux), la sous-évaluation des quantités vient aussi en partie du processus de matching qui répartit la quantité normalement impartie à un ingrédient entre les différents matchs trouvés pour la ligne d'ingrédient. Ou encore si aucun matchs n'a été trouvé dans la base de référence (CIQUAL), la contribution de cette de cette ligne d'ingrédient sera annulée.

2.2 Algorithme Génétique

2.2.1 Introduction

Nous avions envisagé l'approche systématique du fait des contraintes imposées par les règles de composition des nutritionnistes, qui réduisent fortement le champs des possibilités des recettes. Mais cette réduction n'est pas suffisante pour empêcher l'explosion combinatoire lors d'une composition systématique des recettes. Nous devons donc nous tourner vers des algorithmes stochastiques qui sont généralement utilisés pour traiter des espaces de grandes dimensions lorsque l'approche systématique n'est pas adaptée. Malgré tout, l'algorithme systématique nous a permis de nous familiariser avec un certains nombres d'aspects du problème et une partie du code développé a été réutilisée dans les autres algorithmes : la mise au point d'un score de goût des recettes générées, la définition des quantités d'ingrédients (tests avec les quantités GEMRCN ou en utilisant les quantités moyennes des recettes d'Edamam), les règles de composition des nutritionnistes, la compatibilité des règles entre elles (ex : la compatibilité des profils nutritionnels cibles avec les indicateurs holistiques). Notamment, l'un des objectifs de cette exploration était de pouvoir évaluer la couverture nutritionnelle des recettes générées suivant les règles de composition et la possibilité du recouvrement avec les objectifs nutritionnels ou holistiques. La conclusion est qu'il peut être difficile voir impossible de trouver des recettes qui satisfont toutes les contraintes souhaitées (indicateurs holistiques, objectifs nutritionnels des profils...). L'un des avantages des algorithmes stochastiques, comme l'algorithme génétique et l'apprentissage renforcé, est qu'ils permettent de se rapprocher de la solution même si celle-ci est impossible ou difficile.

2.2.2 Vue d'ensemble

Le principe de l'algorithme génétique est de reproduire le procédé naturel de sélection des gènes dans la nature. En effet, l'optimisation procède par sélection des meilleurs recettes suivant des critères nutritionnels et gustatifs parmi une large population d'individus qui se mélangent et subissent des mutations. Le schéma 2.9 présente les grandes étapes du fonctionnement du générateur de recettes utilisant l'algorithme génétique.

2.2.3 Théorie

En analogie à la biologie, on parle de chromosomes pour désigner des séquences de constituants élémentaires appelés gènes. On génère initialement un ensemble de N chromosomes, que l'on appelle population. Suivant les critères d'optimisation recherchés, on définit une "fitness function" qui permet de donner une valeur ou un score à un chromosome. A partir de là, l'algorithme répète les étapes suivantes :

- la sélection de deux chromosomes parents ayant des bons scores parmi les chromosomes de la population.
- l'éventuel mélange des gènes de ces deux chromosomes par croisement de groupe de gènes .
- l'éventuel mutation de certains gènes de ces deux chromosomes.
- la sauvegarde de ces deux chromosomes fils pour pour la constitution d'une nouvelle population.

Lorsque l'on a ainsi reproduit ces étapes N/2 fois, on obtient une nouvelle population constituée à partir de chromosomes de la population initiale ayant les meilleurs scores. Les croisements permettent que des rassemblements fortuits de parties intéressantes de chaque chromosome puissent s'effectuer. Les mutations permettent que des nouveaux gènes ou nouvelles combinaisons de gènes intéressantes soient crées. L'optimisation se poursuit ainsi au fil des générations jusqu'à la satisfaction des critères recherchés (voir [4]).

Pour notre problème de génération de recettes, les chromosomes s'assimilent à des ensembles de quatre plats correspondant chacun à un repas de la journée (petit déjeuner, déjeuner, collation, dîner). Les gènes sont les ingrédients de ces repas. La sous structure supplémentaire qu'est le repas n'a pas de correspondance dans l'algorithme génétique. Bien qu'une journée composée de quatre repas puisse aussi être considérée comme un ensemble d'ingrédients, la sous structure en repas, ainsi que celle en familles d'ingrédients constituant les repas nous obligent à tenir compte de règles de transformations supplémentaires, de façon à ce que les constitutions des journées soient modifiées de façon consistante par l'algorithme génétique. En effet, cer-

taines règles de constitution nous ont été imposées par les nutritionnistes : la constitution des repas à partir d'un certains nombres d'ingrédients issus de certaines familles nutritionnelles (voir 2.2.5), la satisfaction d'objectifs d'apport nutritionnel ou d'évaluation d'indicateurs holistiques à l'échelle d'un repas.

2.2.4 Code Orienté Objet

Notre générateur de recettes utilisant l'algorithme génétique s'implémente naturellement par du code orienté objet. On a définit une classe ingrédient, une classe repas qui se décline en sous-classes "ptitdej", "dejdin", "col" pour petit déjeuner, déjeuner et dîner, et collation. Et puis, on a défini une classe "jour" composée de quatre objets repas, qui eux même sont composés d'une succession d'objets ingrédients. Le schéma 2.10 donne plus de détails sur ces classes, leurs attributs et fonctions.

2.2.5 Règles de composition

Les recettes des repas sont largement déterminées par les "règles de composition" définies par les nutritionnistes de façon à faciliter l'élaboration de recettes équilibrées. Ces règles de composition sont résumées dans le tableau de la figure 2.11. Elles rassemblent les ingrédients en famille sous-divisée en catégories. Le nombre d'ingrédients (composants) par famille et catégorie peut varier entre des limites données suivant les repas. Une certaine quantité d'ingrédients par famille et une certaine répartition entre les catégories sont également imposées par les règles du tableau avec une marge de liberté dans la répartition fixée à environ 20% de la proportion la plus petite. Ainsi, au moment de la création d'un nouveau repas, on commence par déterminer au hasard un nombre d'ingrédients par famille/catégorie (setIngrNb(self,cat)). Puis, famille par famille, suivant le type de repas, on détermine la répartition en poids entre les ingrédients choisis (genFamily(self,fam)). La function getFamProp(self,fam) permet de déterminer une répartition qui fluctue dans la marge autorisée de 20% de la plus petite proportion, en utilisant la distribution de Dirichlet. Lorsqu'une seule catégorie de la famille est présente, la quantité totale de la famille lui est impartie.

2.2.6 Boucle principale : optimisation

Sélection

La sélection s'effectue à chaque fois qu'une nouvelle population est générée. Avant tout, on calcule les probabilités p_i de sélection associées à chaque élément i de la population (jour de quatre repas) par la fonction calcul_probas(population,profil_number,coeff_nutri,coeff_taste). Le calcul du score s_i prend en compte à la fois le score de goût $s^{(t)}$ et le score de nutrition $s^{(n)}$ des éléments. $s^{(t)}$ est relatif à la base de recettes utilisées pour calculer les corrélations entre les ingrédients et $s^{(n)}$ est fonction du profil nutritionnel cible pour lequel on effectue l'optimisation. $\bar{s}^{(t)}$, $\bar{s}^{(n)}$, σ_t, σ_n sont les valeurs moyennes et variances associées aux distributions $\{s_i^{(t)}\}$ et $\{s_i^{(n)}\}$ respectivement. coeff_nutri et coeff_taste, respectivement c_n et c_t dans les equations, permettent d'ajuster l'importance relative de la nutrition et du goût pour le choix des éléments qui vont être retenus pour la génération suivante. On utilise une fonction exponentielle normalisée (softmax) pour le calcul des probabilités :

$$s_i = c_t \frac{\left(s_i^{(t)} - \bar{s}^{(t)}\right)}{\sigma_t} - c_n \frac{\left(s_i^{(n)} - \bar{s}^{(n)}\right)}{\sigma_n} \tag{2.1}$$

$$p_i = \frac{exp(s_i)}{\sum_j exp(s_j))} \tag{2.2}$$

Le tirage s'effectue par la fonction random.choice() de numpy. Le score de nutrition est défini comme l'opposé de la distance de la valeur nutritive des repas aux objectifs nutritionnels fixés par les nutritionnistes. On normalise les valeurs nutritionnelles en divisant par les apports journaliers minimaux recommandés, de façon à uniformiser l'importance relative des nutriments dans le calcul des distances. Le score de nutrition d'un repas est alors obtenu en prenant l'opposé de la somme des valeurs absolues des différences des valeurs nutritives normalisée a leur valeurs minimales et maximales recommandées.

Croisements

Après la sélection de deux jours de la population, on fait un tirage pour déterminer s'ils vont subir un croisement par la fonction crossOver(parents, proba_crossover). Le nombre proba_crossover donne la probabilité qu'un croisement soit effectué. Si c'est le cas, on tire au hasard un repas, puis une famille d'ingrédients de ce repas où la coupure des jours va être effectuée. Ensuite, les repas et ingrédients, avant et après la coupure, sont échangés

entre les deux jours. Il s'agit d'un croisement à un seul pivot (i.e. à une seule coupure) mais nous avons également implémenté des croisements à deux pivots (i.e. coupures) pour échanger des sections plus courtes entres les jours. La nécessité de conserver la structure et la composition des jours en repas/familles pour le respect des règles de composition des nutritionnistes, nous impose d'effectuer des coupures entre les familles. Le but du croisement est de favoriser le rassemblement de parties intéressantes entre deux jours.

Mutations

Une fois l'étape de croisement achevée, la fonction mutate(individu, max_nb_mutations,proba_mutate) détermine si les jours sélectionnés vont subir des mutations en tirant un nombre au hasard et le comparant à proba_mutate. Si une mutation doit avoir lieu on fait un nouveau tirage pour déterminer combien de mutations auront lieu entre une et max_nb_mutations. Une mutation consiste à tirer une famille au hasard parmi les repas du jour. Une nouvelle famille est générée pour remplacer l'ancienne. Encore une fois, le respect des règles de composition des nutritionnistes nous imposent d'effectuer les changements au niveau d'une famille pour pouvoir effectuer le remaniement de la répartition des poids entre les ingrédients de la famille.

2.2.7 Optimisation pour N profils

Jusque là, on a décrit le fonctionnement pour l'optimisation de recettes pour un seul profil nutritionnel. Notre objectif est d'optimiser des recettes pour une population constituée de multiples profils nutritionnels. Pour optimiser les coûts de fabrication de produits alimentaires tout en fournissant des recettes adaptées à chacun, nous avons adopté une stratégie de minimisation du nombres de profils cibles, et une stratégie de minimisation du nombre d'ingrédients utilisés pour les recettes des différents profils cibles.

Agglomération des profils par Hierarchical Clustering

La minimisation des profils cibles s'effectue par agglomération des profils nutritionnels identifiés dans la population à nourrir. Nous employons l'algorithme Hierarchical Clustering, qui constitue progressivement des groupements de proche en proche en évaluant les distances entre profils puis entre groupes de profils. L'avantage du hierarchical clustering est de déterminer lui-même le groupement des profils et le nombre de groupes en fonction d'une distance critique. Cela convient bien à notre problème puisque nous souhaitons réduire le nombre de profils cibles pour satisfaire les besoins nutritifs

de tous par un nombre minimal de recettes. On doit définir une distance critique au delà de laquelle il ne nous est plus permis de grouper les profils, ou autrement dit, il ne nous est plus possible de créer une seule recette pour deux profils éloignés de plus de cette distance. Cela veut aussi dire que nous n'avons pas besoin de créer plusieurs recettes pour des profils qui sont plus proches que cette distance critique. Nous pouvons modifier la métrique qui définit la distance entre les profils en prenant en compte les poids démographiques associés aux profils. Il est normal qu'un profil représentant une part moindre de la population à nourrir soit moins important qu'un profil plus fréquent. Nous utilisons la fonction AgglomerativeClustering de sklearn. On fait la moyenne des profils agglomérés pour obtenir un nombre réduit de profils cibles.

Minimisation des ingrédients à utiliser entre profils cibles, Multiprocessing

L'optimisation requiert une minimisation des ingrédients à utiliser entre les profils de façon à éviter un surcoût pour la réalisation des repas. Pour ce faire, à chaque nouvelle génération, nous faisons l'inventaire des ingrédients des populations par profil puis déterminons les ingrédients communs aux recettes des différents profils. Un nouveau score, score_mini, $s^{(m)}$ (équation 2.3), est ajouté au score de sélection des recettes de la prochaine génération (fonction calcul_probas). Un nouveau coefficient de pondération, coeff_mini (c_m) dans le nouveau score combiné s'_i permet de déterminer le degré d'incitation à la sélection de recettes composées d'ingrédients communs entre les différents profils (equation 2.4).

$$s_i^{(m)} = \frac{\text{nb d'ingredients du menu i communs à tous les profils}}{\text{nb d'ingredients du menu i}} \tag{2.3}$$

$$s'_i = s_i + c_m \frac{\left(s_i^{(m)} - \bar{s}^{(m)}\right)}{\sigma_m} \tag{2.4}$$

Par souci d'efficience, il est préférable de paralléliser les optimisations des menus des différents profils. Le multiprocessing est préférable au multithreading pour plusieurs raisons : nous sommes dans un cas de parallélisme d'un programme du type "CPU-bound" [1] plutôt que "I/O-bound", c'est à dire pour lequel il n'y a pas ou peu de temps morts pendant lesquels le programme est en suspens dû à des transfert de données. D'autre part, le multithreading est en général moins performant en python que le multiprocessing car python a

1. https ://realpython.com/python-concurrency/

été conçu pour exécuter des programmes sur une seule thread [2]. Avec le module multiprocessing, on assigne un processus d'optimisation pour un profil à un coeur du processeur, chaque processus ayant son propre interpréteur python et son propre espace en mémoire. Il n'y a pas de risque de dépassement de la mémoire vive dans notre cas car nous nous limitons au traitement de quelques profils seulement. Nous n'avons pas trouvé de moyen simple de faire synchroniser les processus des différents coeurs afin de déterminer les ingrédients communs aux populations des différents profils. Nous devons donc attendre la terminaison des processus sur chaque coeur (effectuer un "join"), déterminer les ingrédients communs aux différents processus, puis de nouveau relancer l'exécution en parallèle pour continuer l'optimisation (" start" a new "fork"). Comme le module multiprocessing ne permet pas le transfert d'objets imbriqués complexes comme les classes repas ou jours entre les processus, nous avons choisi de faire des sauvegardes temporaires des populations des différents profils à chaque terminaison des processus, et de recharger les populations à chaque redémarrage des processus. Une nouvelle implémentation avec le package Ray [3] devrait permettre de faciliter ces transferts d'objets ainsi que d'améliorer l'efficience de l'algorithme sur plusieurs coeurs.

2.2.8 Novelty Search

"Novelty search" est presenté dans l'article [5]. Nous utilisons cette technique pour favoriser la variété des recettes générées. La création d'une archive de recettes nous permet de sauvegarder les recettes déjà optimisées et de les comparer aux recettes en cours d'élaboration. Après chaque exécution de l'algorithme, les meilleures recettes finales sont ajoutées à l'archive. Un score de nouveauté $s_i^{(nov)}$ (voir équation 2.5), que l'on ajoute au score combiné d'une recette donnée, évalue la rareté de sa composition en regard des recettes déjà archivées. Ainsi nous incitons l'algorithme à sélectionner des recettes de la population qui présentent le plus de nouveauté par rapport aux anciennes recettes. Pour tenir compte de la pertinence d'une recette archivée pour un profil nutritionnel donné, nous avons ajouté un coefficient de pondération défini à partir de la note de compatibilité de la recette avec les profils considérés, $\bar{n}_i$. Cette note, calculée pour chaque repas, est la moyenne des fonctions de Heaviside pour la satisfaction des objectifs par nutriment : un repas dont l'apport en un nutriment est compris entre les bornes min et max des objectifs du profil nutritionnel, reçoit une note de 1 pour ce nutriment et 0 sinon. On fait la moyenne des notes obtenus pour tous les nutriments pour obtenir

2. https ://realpython.com/python-gil/
3. https ://ray.readthedocs.io/en/latest/

la note globale $\bar{n}_i$ pour le repas.

$$s_i^{(nov)} = \frac{1}{k} \sum_{j=0}^{k} \bar{n}_j \frac{1}{d_i(\mu_j)} \tag{2.5}$$

$d_i(\mu_j)$ désigne la distance de Jaccard de la recette i avec les recettes $\{\mu_j\}_j$ de l'archive. La distance de jaccard [4] est une mesure de similarité de deux ensembles, définie comme le rapport du cardinal de l'intersection sur le cardinal de l'union des deux ensembles. Dans notre cas, nous avons voulu substituer les mots-clefs (keywords) de la base d'ingrédients à la place des alim_codes pour l'identification des aliments, afin de donner plus de généralité aux recettes et permettre que des recettes similaires mais composées de variantes des mêmes ingrédients (ex : des farines de différentes qualités) soient considérées comme similaires par le calcul de la distance de Jaccard.

2.2.9 Résultats

Quelques premiers résultats : temps de calculs pour une génération, convergence, résultats des premiers tests de réglage.

Temps de calcul

Population	tps par gen.	nb gen.	prob. mut.	prob. cross.
100	3.5	99	0.1	0.95
100	4.1	50	0.5	0.95
300	12.7	50	0.5	0.95
600	25.6	30	0.5	0.95
1000	44.8	25	0.5	0.95

TABLE 2.1 – Temps de calculs par génération (laptop yoga2 pro, Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz).

Convergence

L'amélioration s'effectue principalement pendant les premières 100 générations. Des exemples de convergence pour le score nutritionnel et pour le score de goût sont montrés pour une population de 300 jours de quatre repas sur les figures 2.13 et 2.14 (l'amélioration correspond à une décroissance pour le score nutritionnel et à une croissance pour le score de goût).

4. https ://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50

Réglage des paramètres

Quels meilleurs paramètres pour l'optimisation ? On a effectué une séries de tests en faisant varier les paramètres de l'algorithme pour déterminer leur influence (voir Figures 2.12).

Population : Quel avantage de l'augmentation de la taille de la population ? Plus la population est grande plus il y a d'ingrédients et de variétés initialement et de possibilités de réarrangements, mais en contrepartie le temps de calcul et le temps de convergence se trouvent allongés. D'après les tests du tableau 2.12, l'augmentation de la population de 250 à 500 jours permet d'augmenter la diversité de la population en terme de nombre recettes (en moyenne 178 recettes pour une population de 500 jours contre 90 pour une population de 250 jours) et d'ingrédients (une moyenne de 144 ingrédients dans la population finale pour une population de 500 jours contre 100 ingrédients en moyenne pour une population de 250 jours). En revanche, les scores obtenus sont en moyenne les mêmes sur 160 itérations pour une population de 250 ou de 500 jours, bien que la convergence ait déjà eu lieu pour les deux populations. Donc il n'est pas forcement avantageux d'augmenter la taille de la population. Plus de tests sont nécessaires pour déterminer la taille optimale qui permette une optimisation rapide et une diversité suffisante.

Proba_crossover (pc), proba_double_pivot (pdp) et proba_mutation (pm) : La proba_crossover détermine la probabilité qu'un croisement ait lieu. D'après les tests du tableau 2.12, une amélioration moyenne du score combiné est observable pour une probabilité de croisement réduite
($exp\_combined = (1.35, 1.33)$ pour $pc = (0.5, 0.95)$) bien que non significative au regard de l'erreur statistique. Si un croisement doit avoir lieu, la proba_double_pivot détermine la probabilité d'un croisement double pivot plutôt qu'un croisement simple pivot ait lieu. On constate une légère amélioration du score combiné en moyenne lorsque l'on utilise le double pivot ($exp\_combined = (1.32 \pm 0.095, 1.36 \pm 0.090, 1.34 \pm 0.13)$ pour $pdp = (0, 0.5, 1)$), bien que non significative en regard de l'erreur statistique. D'après les tests du tableau 2.12 , une augmentation de proba_mutation conduit a une amélioration significative des scores ($exp\_combined = (1.25 \pm 0.04, 1.43 \pm 0.04)$ pour $pm = (0.1, 0.5)$) et à une augmentation de la diversité des populations (en moyenne 76 ingrédients dans les populations finales pour $pm = 0.1$ contre 174 ingrédients pour $pm = 0.5$). Il est donc bénéfique de choisir une valeur relativement élevée de ce paramètre. Em moyenne, toujours d'après ces tests, les meilleurs résultats sont obtenus lorsque $pc = 0.5$, $pm = 0.5$ et

$pdp = 1$.

coeff_nutri (c_n), coeff_taste (c_t) : La convergence du score de nutrition est déplacée vers des meilleurs scores nutritionnels ou vers des meilleurs scores de goût suivant le choix des valeurs relatives des coefficients c_n et c_t. Ceci peut être observé en comparant les résultats sur les figures 2.13 et 2.14 pour $c_n = 2$ et $c_t = 1$ avec les résultats des figures 2.15 et 2.16 quand $c_n = 1$ et $c_t = 2$. Le score de nutrition converge vers une valeur deux fois supérieure lorsque $c_n = 2$ et $c_t = 1$ à celle obtenue lorsque $c_n = 1$ et $c_t = 2$, et inversement pour le score de goût.

Exemples de menus

Les résultats, graphes de convergence et recettes obtenues peuvent être consultés par date dans le répertoire LOG_GA.

2.3 Algorithme d'apprentissage renforcé

La différence entre un algorithme de marche aléatoire (d'exploration) et un algorithme d'apprentissage renforcé (Reinforcement Learning (RL) en anglais) est que l'algorithme d'apprentissage va permettre d'intégrer l'expérience passée (l'exploration) et pouvoir la réutiliser éventuellement lorsque les mêmes situations se représentent (exploitation)[1]. En particulier, pour les espaces d'actions de grande dimensions, les algorithmes de reinforcement learning appelés policy gradient methods sont les plus adaptés [1, 3] (ex : apprendre à jouer à Tétris, maîtriser des mouvements robotiques complexes, choisir le bon ingrédient parmi un large choix ...), et notamment, les derniers algorithmes du type Actor-Critic qui donnent les meilleurs résultats pour ce type de problème (ex :A2C,Advantage Actor Critic [2]).

2.3.1 Apprentissage renforcé - théorie

La théorie de l'apprentissage conceptualise l'interaction d'un agent avec son environnement. Chaque action de l'agent est suivie d'une réponse de l'environnement sous forme d'un retour ou score (reward en anglais) qui mesure la valeur de l'action juste effectuée par l'agent. On intégre l'expérience passée dans des fonctions $\pi(S)$ et $V_\pi(S)$ appelées "policy" et "value function", pour reprendre les termes en anglais, qui permettent ainsi de déterminer l'action à prendre dans un état donné du système [1]. Ces fonctions s'affinent au fur et à mesure de l'exploration. L'environnement est modélisé par une

chaîne de Markov décisionnelle (Markov Decision Process) qui nécessite la définition d'un espace d'état, d'un espace d'action, d'une fonction de score. La figure 2.17 donne un aperçu de l'architecture "Actor-Critic" utilisée. La policy, qui détermine la probabilité d'une action A à un état S donné, est calculée au travers d'un réseau de neurones $fcnn$ de paramètres θ et de la fonction softmax comme suit :

$$\pi\left(A|S,\theta\right) = \frac{exp\left(fcnn\left(S,A,\theta\right)\right)}{\sum_B exp\left(fcnn\left(S,B,\theta\right)\right)} \tag{2.6}$$

L'entrainement du réseau de neurones s'effectue par gradient descent par la formule d'actualisation suivante ([1],chapitre 13) :

$$\theta_{t+1} = \theta_t + \alpha\left(-V_\omega(S_t)\right)\nabla_\theta ln\pi\left(A_t|S_t,\theta_t\right) \tag{2.7}$$

qui peut se réécrire :

$$\theta_{t+1} = \theta_t + \alpha\left(R_t + \gamma V_\omega\left(S_{t+1}\right) - V_\omega\left(S_t\right)\right)\nabla_\theta ln\pi\left(A_t|S_t,\theta_t\right) \tag{2.8}$$

$$\theta_{t+1} = \theta_t + \alpha\delta_t\nabla_\theta ln\pi\left(A_t|S_t,\theta_t\right) \tag{2.9}$$

La value function $V_\omega(S_t)$ est la moyenne de la q-value function $Q_\omega(S_t,A_t)$. Ainsi, l'entraînement du réseau de neurones de la policy nécessite l'entraînement d'un deuxième réseau de neurones de paramètres ω déterminant $Q_\omega(S_t,A_t)$ et $V_\omega(S_t)$. Ce deuxième réseau est appelé "critic" alors que le réseau déterminant la policy est appelé "actor". δ_t est appelé l'Avantage et représente à quel point il est avantageux de choisir l'action A_t par rapport aux autres actions, et donne son nom Advantage Actor Critic à l'algorithme.

En outre, cet algorithme est parallélisable en lançant plusieurs agents explorant l'environnement en même temps et utilisant les même paramètres θ et ω pour leur réseaux.

L'apprentissage renforcé profond (deep reinforcement learning) consiste à entraîner ces réseaux de neurones pour déterminer la bonne action à prendre en fonction de l'état de l'environnement.

Pour notre problème, l'état du système est défini par la donnée du menu de la recette, c'est à dire des ingrédients qui le compose, ainsi que de sa valeur nutritionnelle renseignée par nutriments. Il constitue un tableau d'environ 42 paramètres qui sont normalisés pour le réseau de neurones. L'espace des actions est défini par l'ensemble des changements possibles concernant le menu, ce qui revient à faire un nouveau choix pour la composition d'ingrédients. Le Score (Reward) est obtenu par une combinaison de la distance de la recette aux objectifs nutritionnels et d'un score de goût comme pour l'algorithme génétique. L'espace d'état, l'espace d'action et le score sont défini dans l'environnement OpenAI Gym du programmes AIRecipes.py.

2.3.2 Choix du framework : OpenAI Baselines, OpenAI Stable-baselines ou RLlib

L'utilisation de librairies d'apprentissage renforcé open source pour réaliser nos programmes est préconisée pour assurer l'efficience et la fiabilité des programmes testés par une large communauté. Mais, c'est aussi un garde fou et cela permet d'utiliser des solutions qui ont fait leur preuve. Cela permet également de tester facilement différente version d'algorithmes disponibles pour déterminer celui qui marche le mieux. En revanche, l'utilisation de librairies existantes nécessite de se conformer à certaines contraintes et limitations de la librairie. En fonction de l'état de développement de la librairie, certaines solutions n'ont pas forcément été encore implémentées. Notamment, un problème auquel nous avons été confronté fut la définition d'un espace d'action adapté à notre problème. Idéalement nous avions besoin d'un espace d'actions paramétré (parametrized action space) ou éventuellement d'un espace d'action hiérarchisé (hierarchical action space), qui permettraient de choisir certains ingrédients en certaines quantités. Malheureusement, ce types d'espace d'action ne sont pas encore implémenté de façon standard dans les librairies open source d'apprentissage renforcé. Nous avons utilisé tour à tour OpenAI Gym [5], OpenAI Gym stable-baselines [6], et RLlib [7]. OpenAI a été fondé initialement pour aider le développement et la diffusion de la technologie AI, mais la documentation n'est pas bien fournie et le code open source pas facile à utiliser. En outre, la compagnie semble avoir dévié de son but original et a été privatisée récemment. Une branche des programmes originaux a été développé initialement par un groupe de l'Inria/Ensta ParisTech qui a poursuivi le développement dans l'esprit original en fournissant une documentation bien maintenue et détaillée. Malgré cela, ayant rencontré des difficultés dans un premier temps avec OpenAI Stable baselines pour l'utilisation de A2C en multiprocessing et étant intéressé par la possibilité d'utiliser des espaces d'action plus adaptés à notre problème non disponibles sur OpenAI stable baselines (parametrized action space ou hierarchical action space), nous nous sommes tournés vers la librairie RLlib qui était recommandée sur les forums et offraient plus de possibilités que openAI stable baselines de ce point de vue. Puis, dans un troisième temps, ayant également rencontré des difficultés avec RLlib et avant de se lancer dans l'utilisation d'espaces d'actions plus complexes, nous avons voulu effectuer des tests dans le but de valider l'approche RL pour la résolution de notre problème sur une modélisation plus simple qui pouvait être implémentée avec la librairie OpenAI

5. https ://github.com/openai/baselines
6. https ://github.com/hill-a/stable-baselines
7. https ://ray.readthedocs.io/en/latest/rllib.html

stable-baselines.

2.3.3 Environnement Gym

Toutes les librairies utilisent, ou sont compatibles, avec le toolkit Gym d'OpenAI pour la définition de l'environnement. La définition de l'environnement passe par la définition d'un espace d'état (ou d'observation), d'un espace d'action et des fonctions step(), reset(), et render(). L'espace [8] peut être un tuple ou dictionnaire d'espaces plus simples, mais en pratique les librairies ne permettent pas de gérer des espaces complexes directement avec les modèles implémentés. Pour cette raison, nous avons choisi de modéliser l'espace d'état avec un espace Box [9], c'est à dire un tuple de paramètres variant continûment dans l'intervalle $[0, \infty]$, et représentant le menu à l'instant t, codé par les indices renormalisés des aliments dans la base d'aliments et les valeurs nutritionnelles renormalisées pour chaque nutriment, soit 42 paramètres au total pour un menu d'environ 10 ingrédients. Pour l'espace d'action, nous avons choisi de discrétiser les possibilités de choix de quantités pour pouvoir utiliser un espace d'action multi_discrete [10] qui permet de choisir un ingrédient en trois variantes de poids (min,moyen,max) pour chacune des catégories prédéfinies du menu. La fonction step() de l'environnement Gym est la plus importante. Elle définit l'effet d'une action sur l'environnement i.e. sur l'espace d'observation, et le reward obtenu en réponse de l'environnement. la fonction reset() permet de redémarrer l'environnement et est appelée à chaque réinitialisation de l'environnement. La fonction render() permet de visualiser l'environnement et son évolution. Pour nous, cela consiste à afficher les recettes et leurs valeurs nutritionnelles.

2.3.4 Résultats

L'algorithme permet actuellement d'optimiser des recettes mais n'apprend pas réellement. La convergence s'effectuera vers la même recette optimisée. Il faudrait introduire une intégration de l'expérience sur les recettes obtenues après optimisation pour engendrer un apprentissage à ce niveau là au fur et a mesure de l'élaboration de nouvelles recettes. L'utilisation d'un espace d'action hiérarchique [11] pourrait être approprié.

8. https ://github.com/openai/gym/tree/master/gym/spaces
9. https ://github.com/openai/gym/blob/master/gym/spaces/box.py
10. https ://github.com/openai/gym/blob/master/gym/spaces/multi_discrete.py
11. https ://thegradient.pub/the-promise-of-hierarchical-reinforcement-learning/

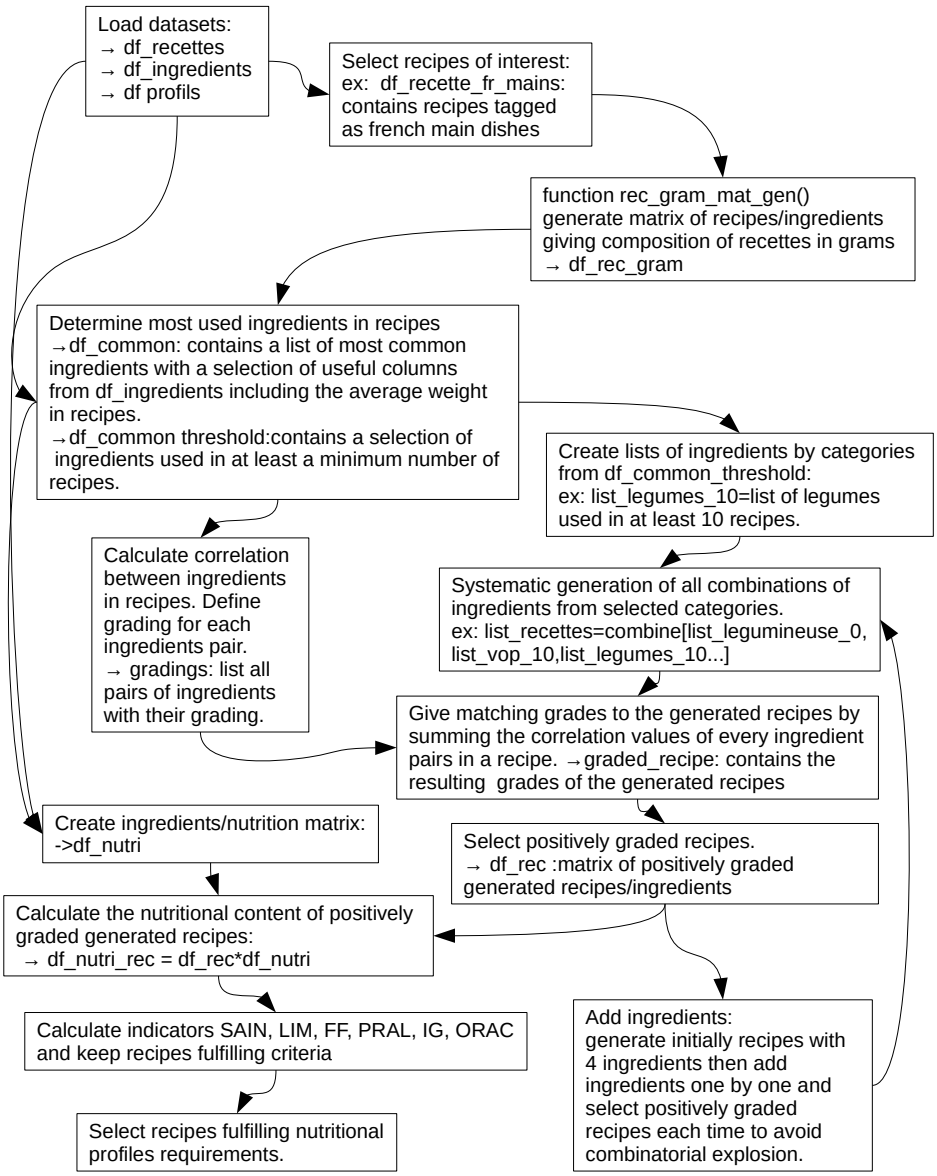

FIGURE 2.1 – Schéma de fonctionnement du générateur systématique pour des recettes de plats principaux français

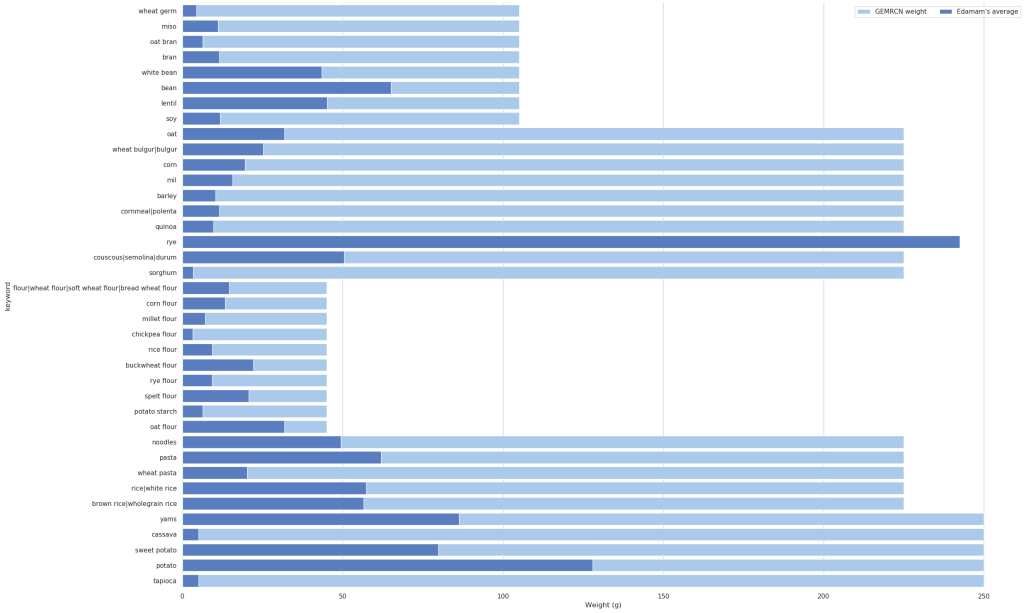

FIGURE 2.2 – Comparaison des quantités moyennes de féculents des plats principaux français (french main dishes) en bleu foncé aux quantités recommandées par GEMRCN en bleu clair.

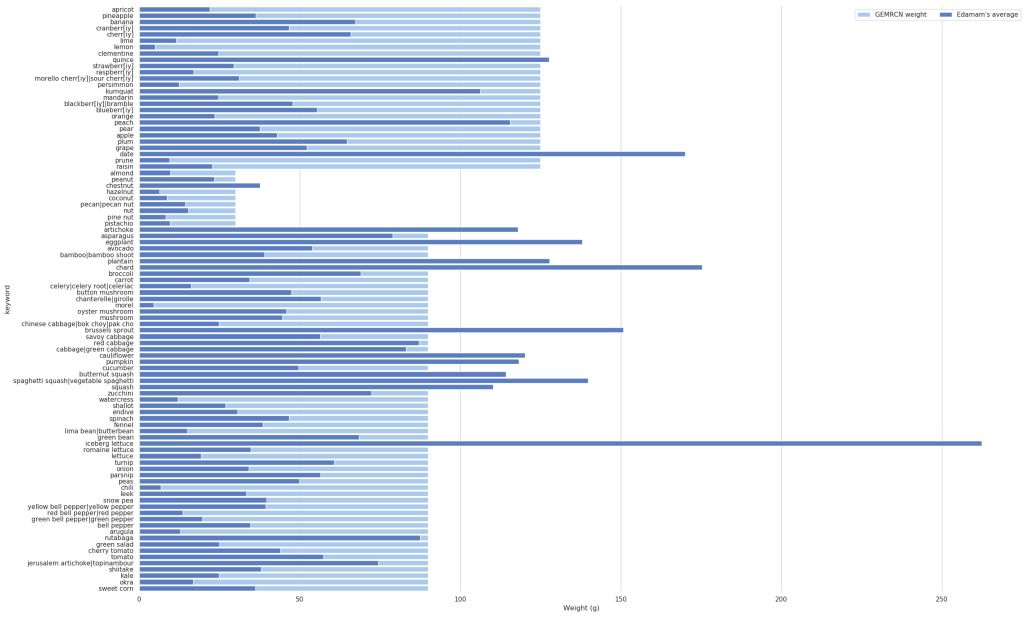

FIGURE 2.3 – Comparaison des quantités moyennes de fruits et légumes des plats principaux français (french main dishes) en bleu foncé aux quantités recommandées par GEMRCN en bleu clair.

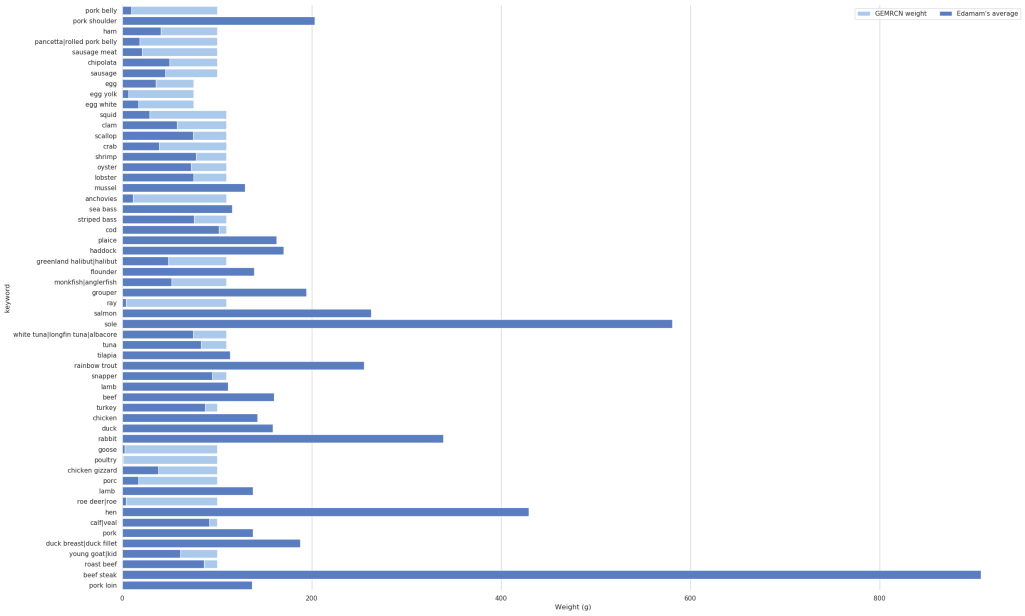

FIGURE 2.4 – Comparaison des quantités moyennes des viandes, oeufs, poissons des plats principaux français (french main dishes) en bleu foncé aux quantités recommandées par GEMRCN en bleu clair.

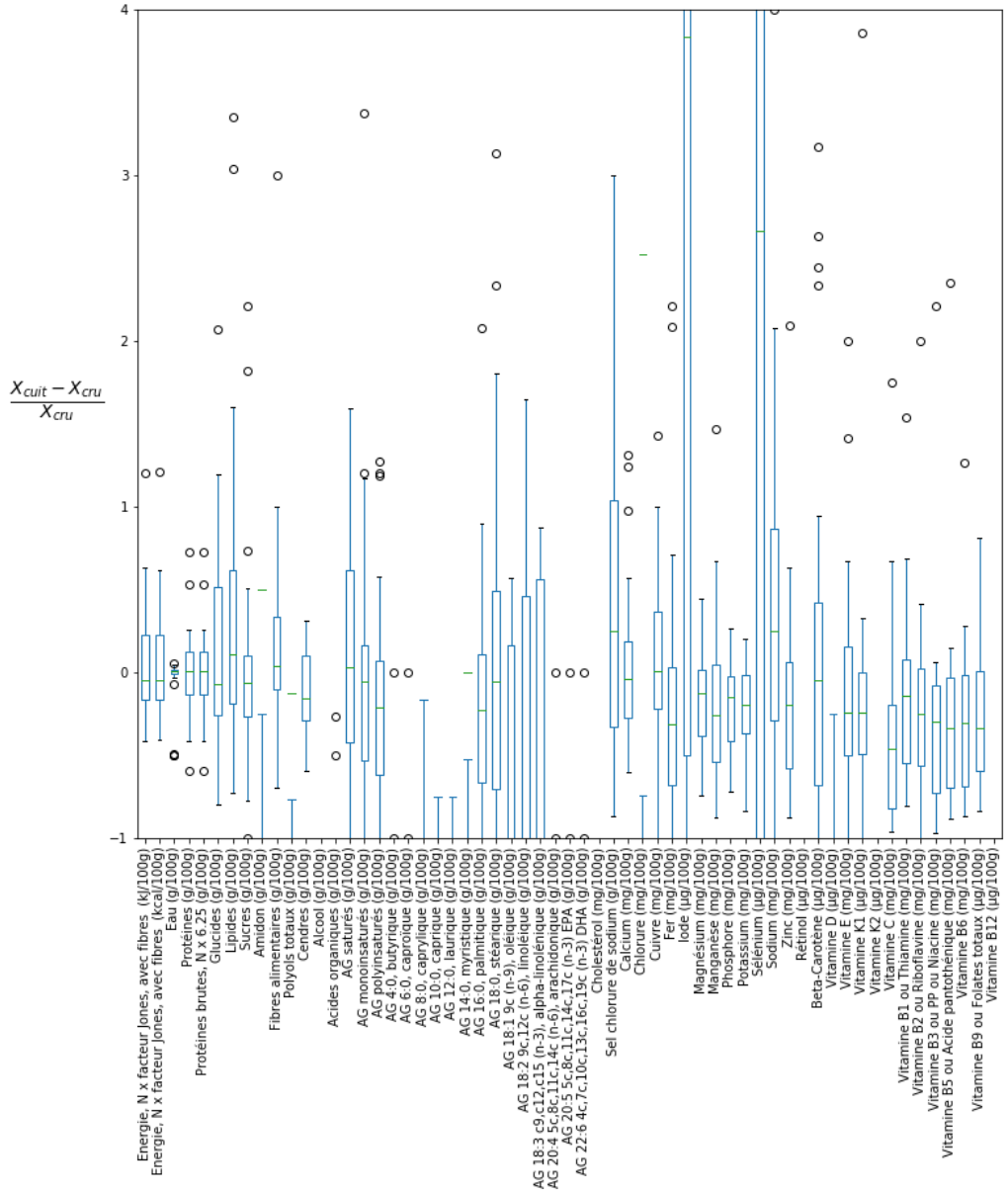

FIGURE 2.5 – Effet de la cuisson sur les légumes.

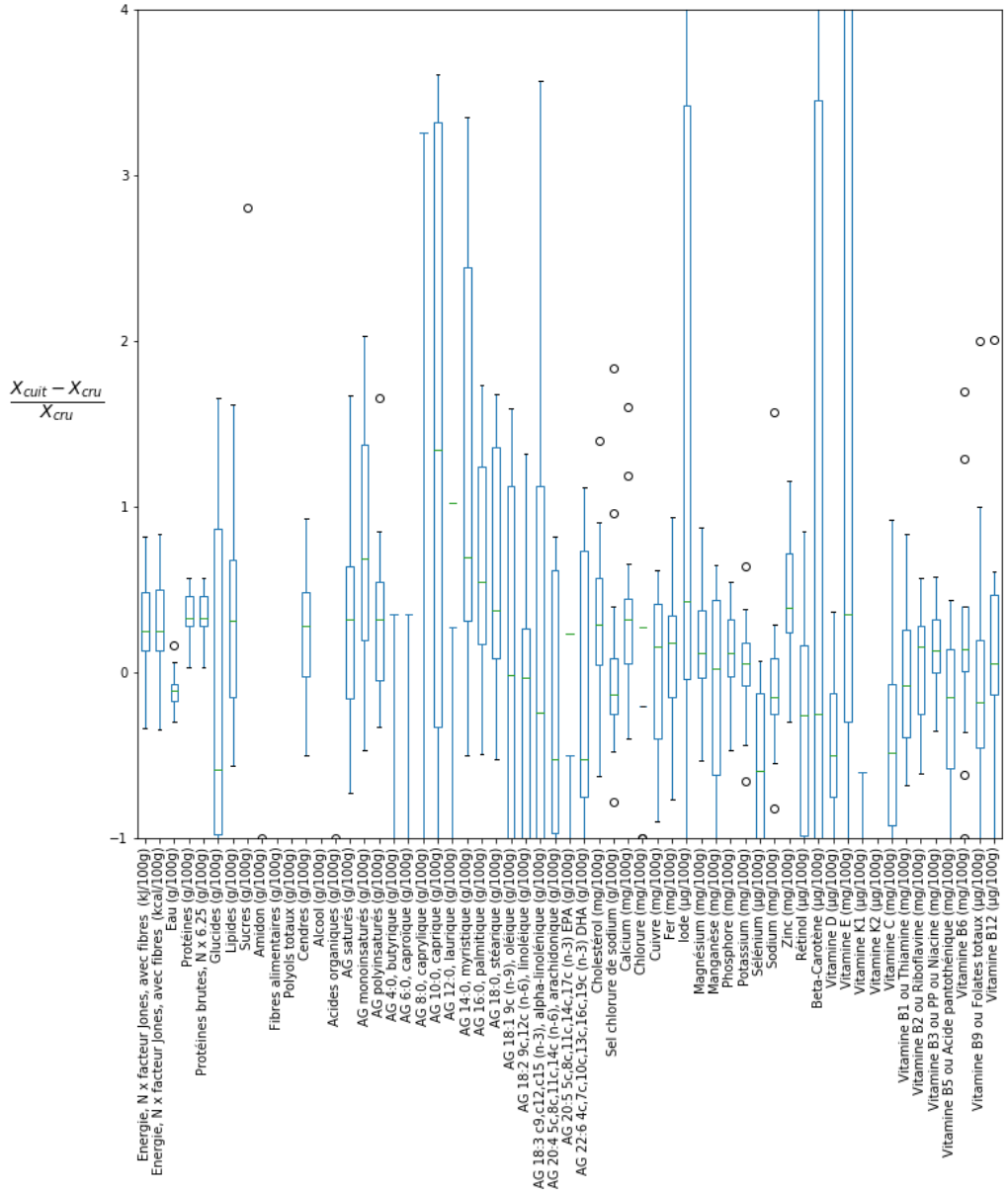

FIGURE 2.6 – Effet de la cuisson sur les viandes.

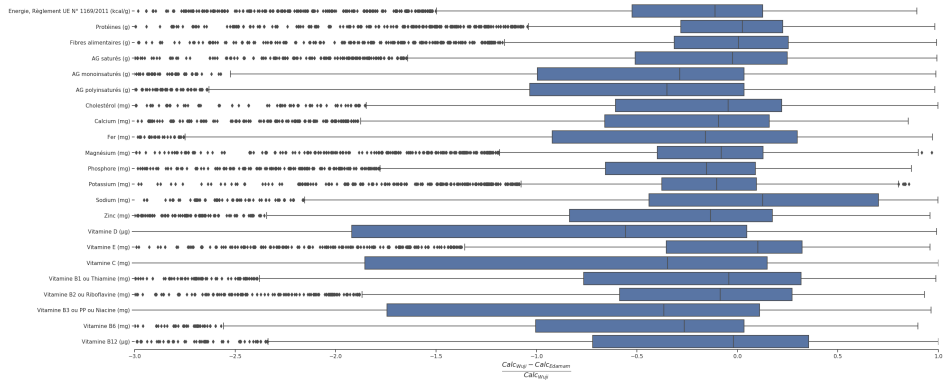

FIGURE 2.7 – Comparaison de nos calculs nutritionnels pour les recettes d'Edamam de plats principaux français avec les valeurs nutritionnelles fournies par Edamam pour les mêmes recettes.

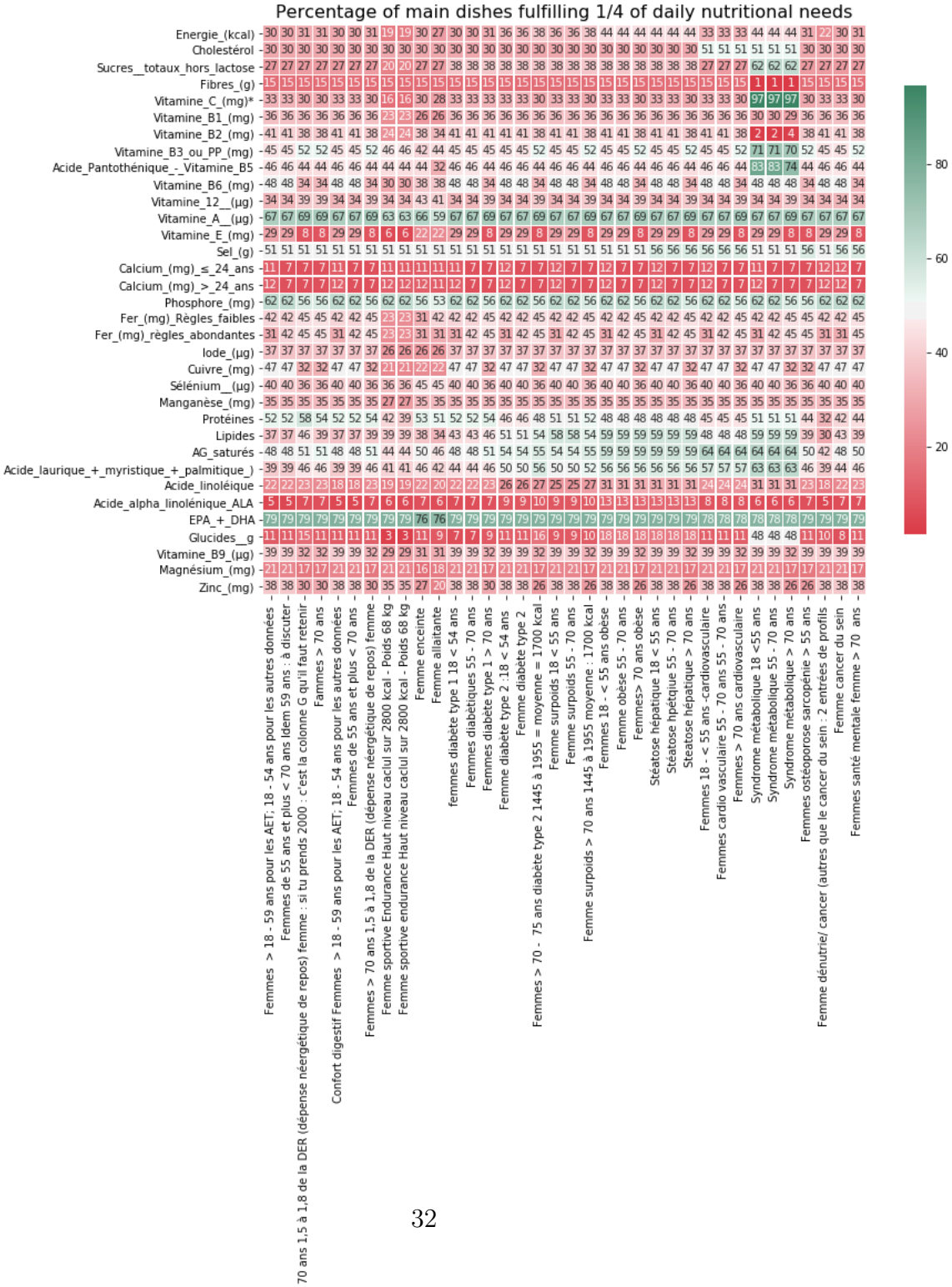

FIGURE 2.8 – Edamam's french main dishes nutrition

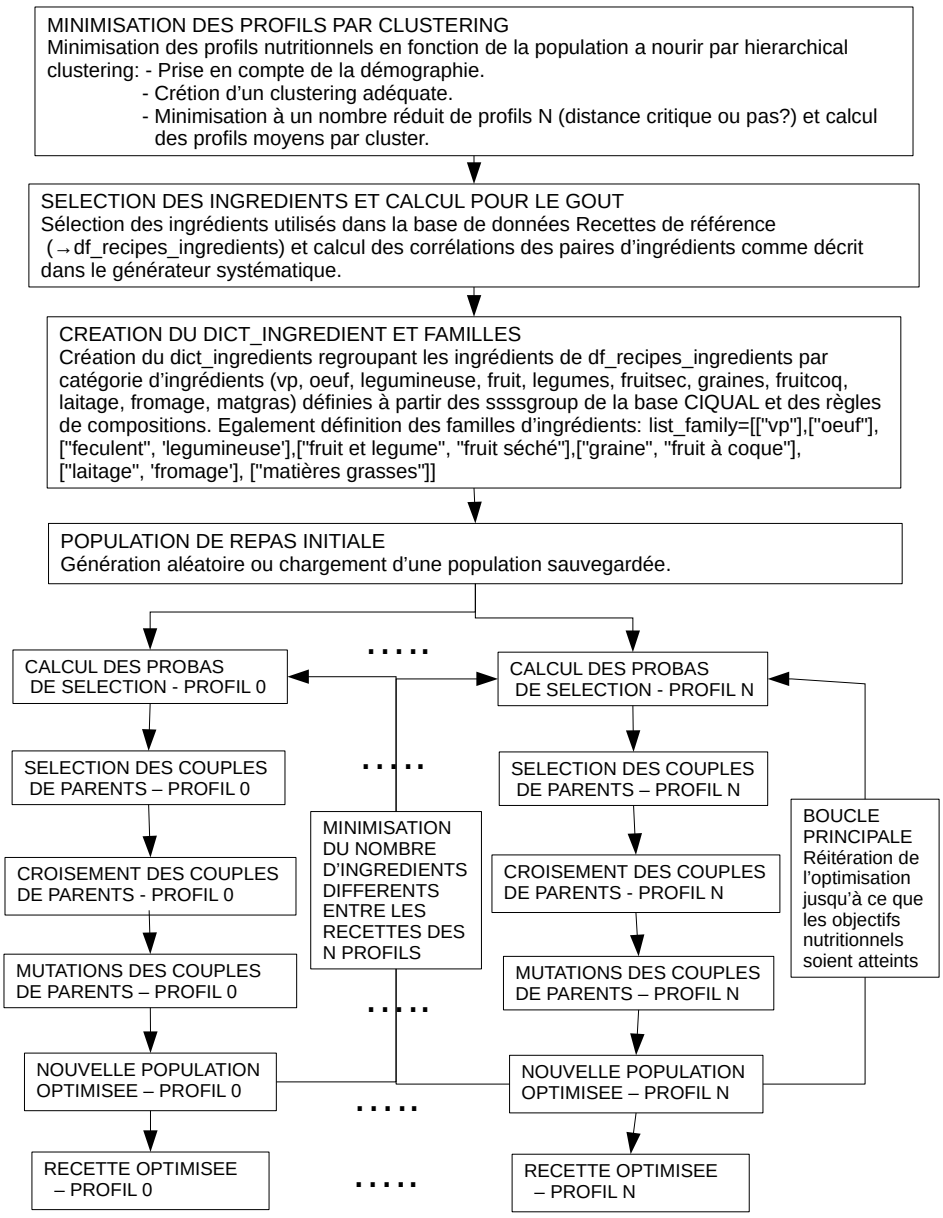

FIGURE 2.9 – Schéma de fonctionnement de notre programme genetic.py générateur de recettes utilisant l'algorithme génétique.

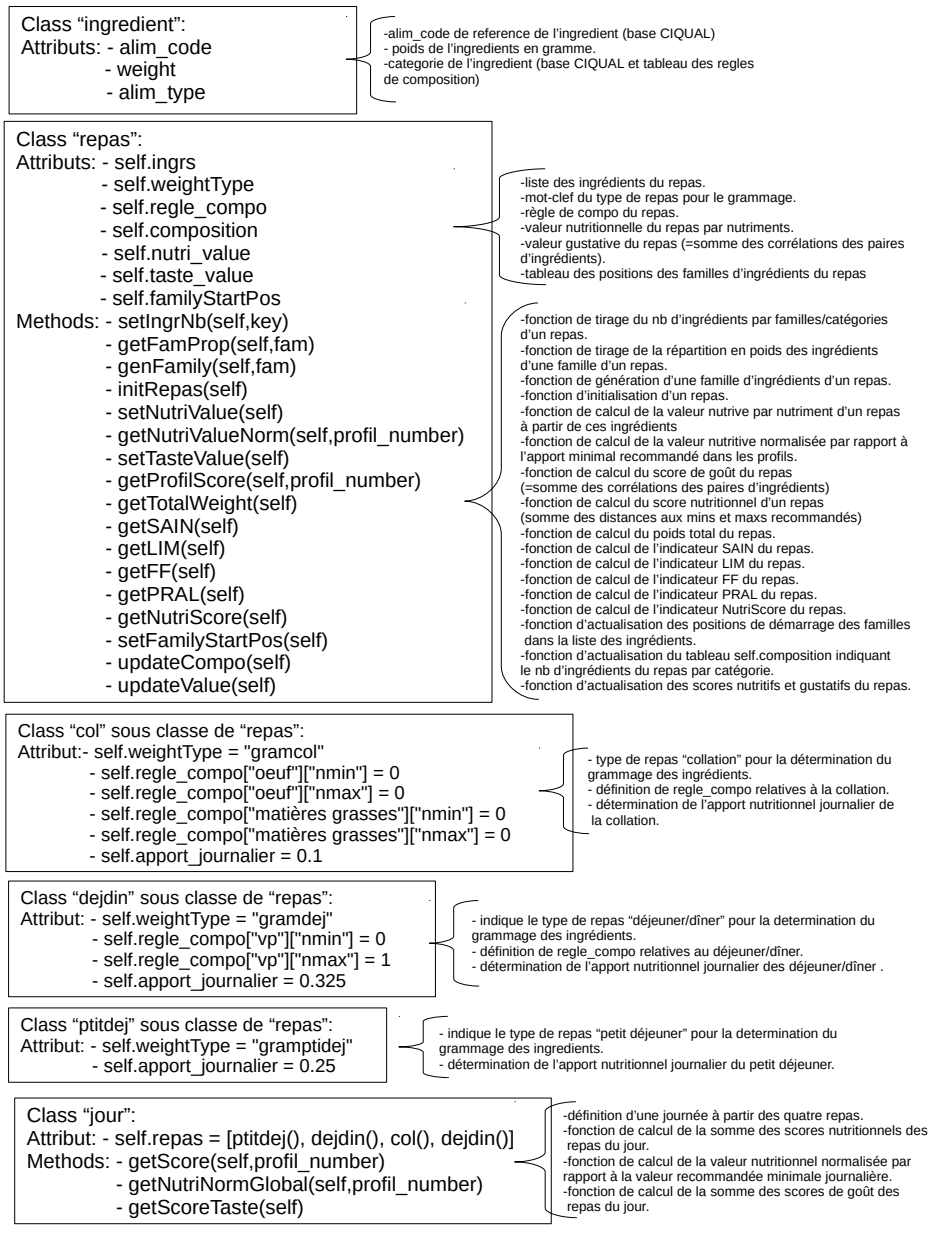

FIGURE 2.10 – Classes du programme genetic.py et leur descriptions.

5% de marge d'erreur sur le grammage total			marge d'erreur 20%
Petit-déjeuner	**Déjeuner/Dîner**	**Collation**	Cible
			30%
150	200	100	**70%**
			85%
100	250	100	15%
20	20	20	50%
			50%
			20%
150	150	100	**80%**
15	15	0	**100%**
0	100	0	**100%**
50		50	**100%**

FAMILLE	CATEGORIE	COMPOSANTS
Aides culinaires	plantes	+++
Féculents	légumineuses	
	complément céréaliers	**2 à 3**
	pâtes, riz et céréales	
	pommes de terre et autres tubercules	
Fruits et légumes	**fruits**	**2 à 3**
	légumes	
	fruits séchés	**0 à 1**
Graines oléagineux	graines	**1 à 2**
	fruits à coque	
Laits et produits laitiers	fromages	
	laits	**1 à 2**
	produits laitiers frais et assimilés	
Matières grasses ajoutées	**animale**	**0 à 1**
	végétale	
Viandes Poissons	**charcuteries**	
	poissons	**0 à 1**
	viandes crues	
	viandes cuites	
Œufs	Œufs	**0 à 1**

FIGURE 2.11 – Règles de composition des repas.

pop	pc	pdp	pm	cn	ct	ni	mean exp_nutri	mean exp_taste	mean exp_combined	best exp_nutri	best exp_taste	best exp_combined	nb ingrs pop.	nb rec. diff.
250	0.95	0	0.1	1	0.5	160	1.18	1.07	1.21	0.63	2.31	1.46	72	31
250	0.95	0	0.5	1	0.5	160	1.24	1.09	1.4	1.72	1.25	2.14	142	123
250	0.95	0.5	0.1	1	0.5	160	1.18	1.06	1.29	1.71	0.87	1.49	62	21
250	0.95	0.5	0.5	1	0.5	160	1.26	1.09	1.39	2.24	1.3	2.92	142	165
250	0.95	1	0.1	1	0.5	160	1.17	1.12	1.26	0.75	3.18	2.38	68	39
250	0.95	1	0.5	1	0.5	160	1.26	1.11	1.46	2.1	1.56	3.27	146	157
250	0.5	0	0.1	1	0.5	160	1.58	1.07	1.26	1.94	0.97	1.88	61	31
250	0.5	0	0.5	1	0.5	160	1.33	1.13	1.39	1.87	1.58	2.95	160	159
250	0.5	0.5	0.1	1	0.5	160	1.21	1.09	1.29	1.51	1.24	1.87	64	27
250	0.5	0.5	0.5	1	0.5	160	1.29	1.09	1.43	1.78	1.82	3.23	157	153
250	0.5	1	0.1	1	0.5	160	1.26	1.11	1.23	0.62	2.8	1.73	61	40
250	0.5	1	0.5	1	0.5	160	1.29	1.1	1.5	2.15	1.49	3.2	143	134
500	0.95	0	0.1	1	0.5	160	1.12	1.12	1.19	0.7	4.4	3.08	86	65
500	0.95	0	0.5	1	0.5	160	1.34	1.12	1.42	2.37	1.11	2.64	194	271
500	0.95	0.5	0.1	1	0.5	160	1.15	1.07	1.22	1.18	1.18	1.39	82	58
500	0.95	0.5	0.5	1	0.5	160	1.27	1.1	1.46	1.67	1.78	2.98	193	252
500	0.95	1	0.1	1	0.5	160	1.17	1.06	1.25	1.92	0.85	1.62	72	39
500	0.95	1	0.5	1	0.5	160	1.21	1.09	1.36	1.7	1.29	2.2	186	273
500	0.5	0	0.1	1	0.5	160	1.18	1.05	1.27	1.15	1.94	2.23	87	55
500	0.5	0	0.5	1	0.5	160	1.4	1.14	1.4	1.63	2	3.27	205	355
500	0.5	0.5	0.1	1	0.5	160	1.21	1.07	1.32	1.2	1.58	1.89	102	75
500	0.5	0.5	0.5	1	0.5	160	1.28	1.11	1.46	1.58	0.09	3.28	215	306
500	0.5	1	0.1	1	0.5	160	1.24	1.14	1.18	0.97	1.85	1.8	99	86
500	0.5	1	0.5	1	0.5	160	1.32	1.1	1.49	2.4	1.45	3.49	209	305

FIGURE 2.12 – Table de tests des paramètres de l'algorithme : les scores exponentiés moyens et ceux de la meilleure recette sont indiqués, ainsi que le nombre d'ingrédients différents et le nombre de recettes différentes de la population finale (après 160 itérations).

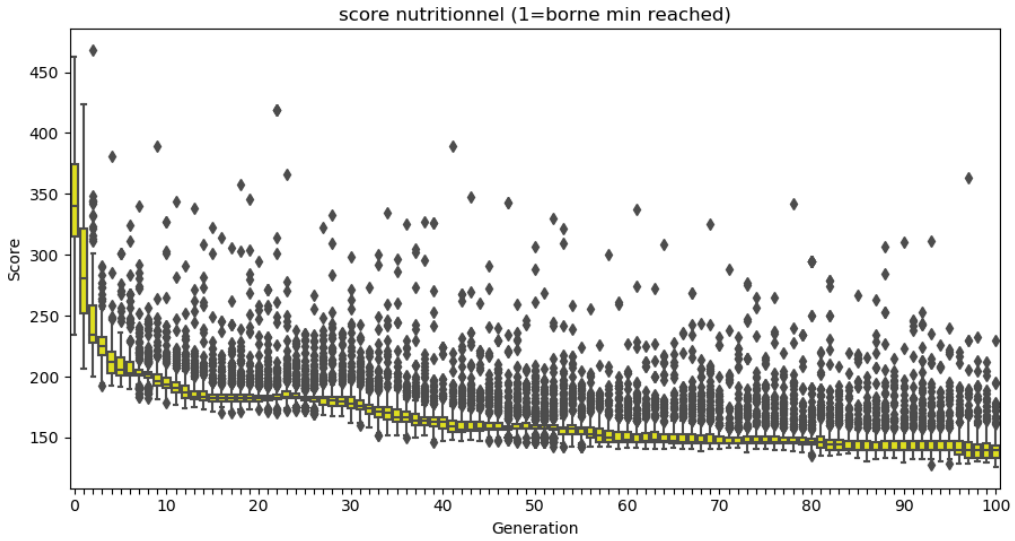

FIGURE 2.13 – Convergence du score nutritionnel pour une population de 300 jours de quatre repas initialement générée aléatoirement (proba_crossover = 0.75, proba_mutate = 0.4, coeff_nutri = 2, coeff_taste = 1).

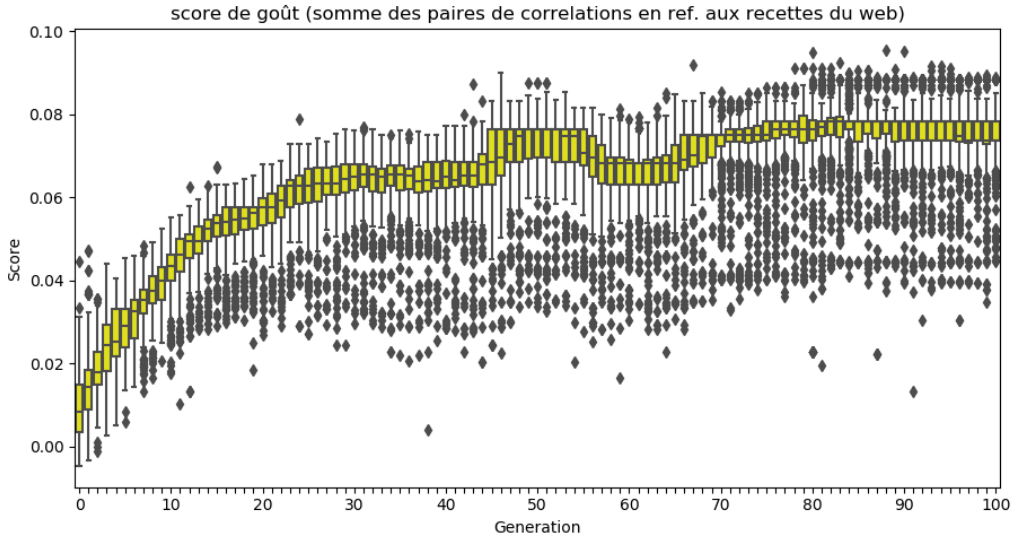

FIGURE 2.14 – Convergence du score de goût pour une population de 300 jours de quatre repas initialement générée aléatoirement (proba_crossover = 0.75, proba_mutate = 0.4, coeff_nutri = 2, coeff_taste = 1).

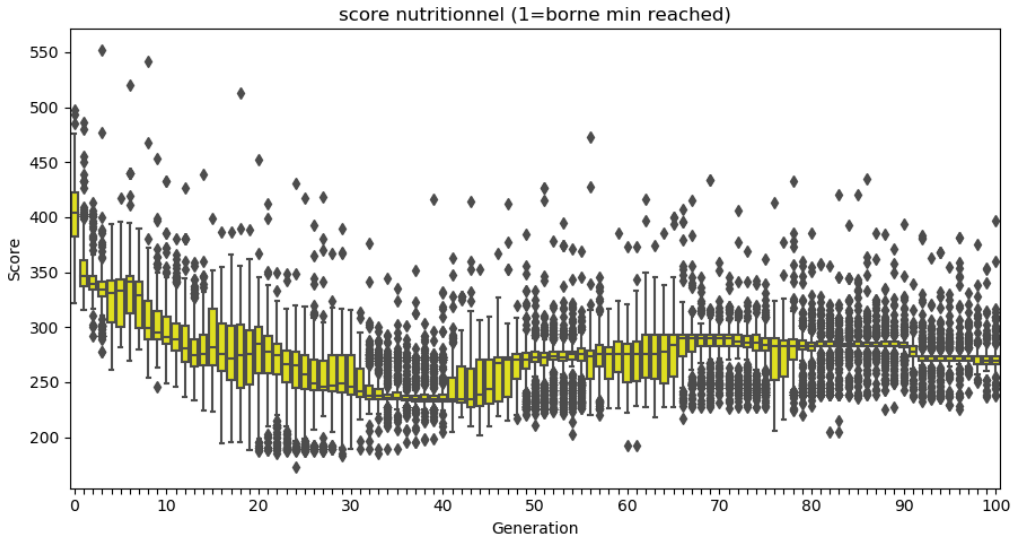

FIGURE 2.15 – Convergence du score nutritionnel pour une population de 300 jours de quatre repas initialement générée aléatoirement (proba_crossover = 0.75, proba_mutate = 0.4, coeff_nutri = 1, coeff_taste = 2).

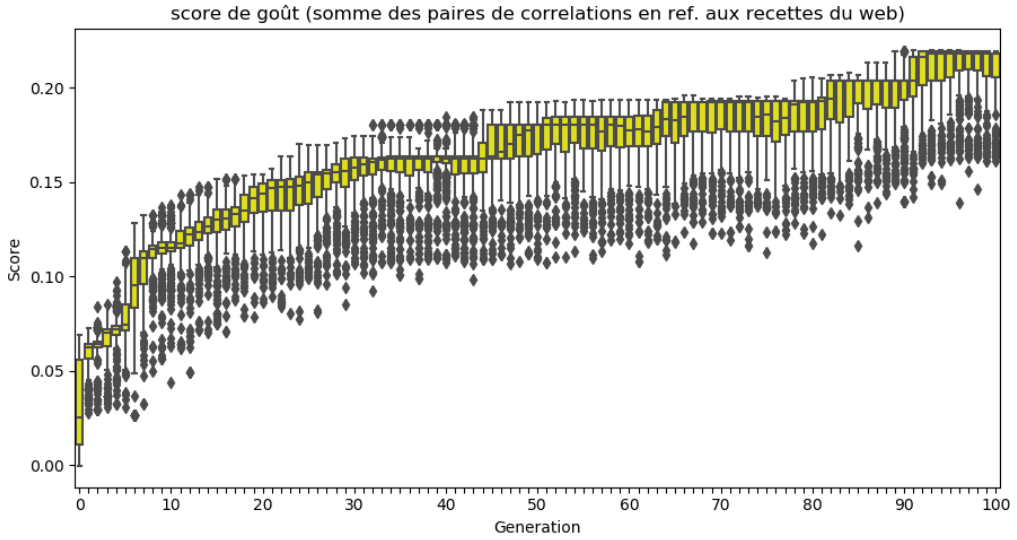

FIGURE 2.16 – Convergence du score de goût pour une population de 300 jours de quatre repas initialement générée aléatoirement (proba_crossover = 0.75, proba_mutate = 0.4, coeff_nutri = 1, coeff_taste = 2).

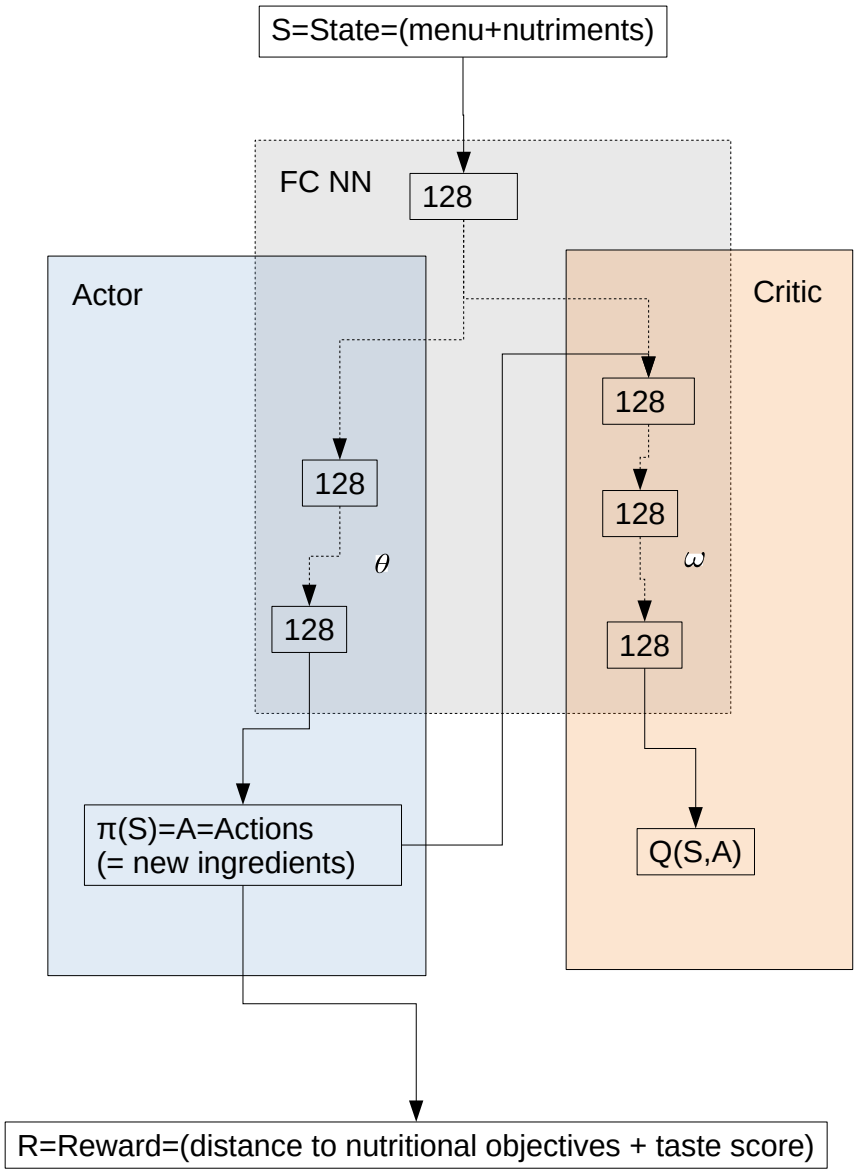

FIGURE 2.17 – Schéma explicatif de l'architecture Actor-Critic de l'algorithme d'apprentissage renforcé.

Chapitre 3

Conclusions

Pendant une dizaine de mois consacrés au développement de cet outil de formulation pour Wuji & co, nous avons exploré différentes approches algorithmiques. Le premier générateur, surnommé générateur systématique, nous as permis de nous familiariser avec le problème et d'envisager une solution "brute force". Cette voie n'a pas aboutie du fait des difficultés de contourner le problème d'explosion combinatoire liée à l'examen systématique des combinaisons possibles d'ingrédients. Plus de temps aurait été nécessaire pour rendre une conclusion finale concernant cette approche, nous avons préféré nous tourner vers des approches algorithmiques stochastiques plus prometteuses. Parmi ces approches l'approche algorithme génétique est la plus naturelle pour le problème de génération de recettes et la plus simple à implémenter. Nous avons développé une approche classique basée sur une recherche d'optimisation par rapport à des objectifs (nutritionnels, gustatifs,...). Cette approche nous a permis d'obtenir des recettes optimisées satisfaisant le cahier des charges et les règles de composition des nutritionnistes. Nous pensons que l'utilisation d'un score de goût basé sur les corrélations des ingrédients dans les recettes de la base de recettes, ne donne pas le meilleur résultat possible, et que celui-ci devrait être remplacé par un score de goût déterminé à partir de l'évaluation de la proximité contextuelle des ingrédients en utilisant Word2Vec. D'autre part, les règles de composition doivent être revues pour éviter d'imposer le mélange inapproprié de certaines catégories d'ingrédients comme les fruits et légumes. Il est également nécessaire de permettre plus de flexibilité dans la composition des recettes pour permettre aux recettes de se rapprocher des objectifs nutritionnels, notamment en ce qui concerne la variations des quantités. Une autre approche, utilisant le même algorithme génétique de base, pourrait être mise en place pour réaliser une exploration de l'espace des recettes par le biais de la technique "novelty search", qui se contente de sélectionner les recettes pour leur caractère novateur. Pour

l'instant, nous avons uniquement utilisé le principe de "novelty search" en combinaison avec une recherche multi-objectives, pour favoriser la génération de recettes différentes de celles déjà archivées. L'application de cette technique pour une exploration par pure recherche de nouveauté pourrait donner des résultats intéressants et devrait être essayer. L'algorithme d'apprentissage renforcé, d'abord retenu dans la perspective de réaliser un algorithme d'apprentissage de la formulation au delà de la simple recherche stochastique, s'est avéré plus difficile à implémenter et tout compte fait moins naturel. Malgré tout, en simplifiant les objectifs de génération de recettes, nous avons réussis à mettre en place un algorithme qui permette d'optimiser des recettes mais n'apprend pas réellement. Il faudrait ajouter un niveau supérieur d'intégration de l'expérience pour faire en sorte qu'il apprenne à s'améliorer à partir des résultats des recettes optimisées. Plus de travail est nécessaire pour déterminer les possibilités réelles de cet algorithme.

Bibliographie

[1] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning :An Introduction*, The MIT Press Cambridge, Massachusetts London, England, 2nd edition, 2018.

[2] Volodymyr Mnih et al (Google Deepmind) *Asynchronous Methods for Deep Reinforcement Learning.* arXiv :1602.01783, 2016.

[3] Gabriel Dulac-Arnold, Richard Evans et al (Google Deepmind) *Deep Reinforcement Learning in Large Discrete Action Space.* arXiv :1512.07679, 2016.

[4] Richard S. Sutton and Andrew G. Barto Stephan Marsland, *Machine Learning, An algorithmic perspective*, CRC Press, Taylor & Francis Group, 2nd edition, 2015.

[5] Joel Lehman and Kenneth O. Stanley *Evolving a Diversity of Virtual Creatures Through Novelty Search and Local Competition* Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2011).