# Rapport du projet "Emploi du temps"

BALZANO Antoine, LAULLIER Hugo

24 Janvier 2021

# Sommaire

# 1 Introduction

Dans ce projet, nous avons été amené à implémenter un algorithme permettant de créer des emplois du temps pour une université selon certaines contraintes. Afin de résoudre ce problème d'optimisation, nous avons utilisé un langage spécialisé dans l'évolution artificielle : EASENA.

# 2 Organisation du projet

## 2.1 Différents fichiers et dossiers

Voici ce que vous pourrez retrouver dans les différents dossiers et fichiers constituants notre projet :

- Dans le dossier include se trouve la librairie rapidjson, qui permet de lire facilement les fichiers JSON.

- Dans le dossier json se trouvent deux dossier : in (pour les fichiers d'entrées) et out (pour les fichiers de sortie). Dans le dossier in, le fichier data.json correspond aux paramètres de l'université nécessaire à la création des emplois du temps (formations, filières, professeurs, salles de cours...). Dans le dossier out, le fichier res.json correspond aux différents emplois du temps proposés afin de répondre à un maximum de contraintes, et le fichier penalities.json liste combien de contraintes n'ont pas été respectées selon leur importance.

- Le fichier main.js permet l'affichage final des emplois du temps, et le fichier index.html correspond au code de la page web sur laquelle seront afficher les emplois du temps finaux.

- Le fichier schedule.ez comporte l'algorithme évolutionnaire permettant la création des emplois du temps.

- Le fichier seeds.txt comporte quelques graines offrant des résultats intéressants.

- Le fichier rapport.pdf est ce document, et décrit dans les détails ce projet.

## 2.2 Prérequis

Afin que le projet fonctionne, il est nécessaire que vous installez EASENA. Vous pouvez retrouver la procédure d'installation via ce lien : `http://easea.unistra.fr/index.php/Installing_EASEA`. Afin d'afficher les emplois du temps finaux, vous avez aussi besoin d'installer php sur votre machine.

## 2.3 Utilisation

Une fois EASENA et php installés, voici les étapes permettant de faire fonctionner correctement le projet :

- Dans le dossier schedule.ez, des lignes 18 à 24, ajustez les poids des contraintes à votre convenance. Vous pouvez garder ceux proposés par défaut, ils fonctionnent.

- Dans le fichier data.json (qui se trouve dans le dossier json/in/), ajustez les paramètres relatifs à votre université (formations, filières, professeurs, salles de cours...).

- Dans le dossier schedule.ez, des lignes 806 à 835, ajustez les hyperparamètres de l'algorithme à votre convenance. Vous pouvez garder ceux proposés par défaut, ils fonctionnent.

- Effectuez la commande easena schedule.ez : vous obtenez un Makefile généré par easena.

- Lancer la commande make : un fichier exécutable schedule est ainsi créé.

- Faites ./schedule : vous lancez l'algorithme évolutionnaire. Vous avez à disposition un graphique dynamique qui permet de suivre l'évolution du programme.

- A la fin de l'exécution, vous pourrez retrouver un affichage écrit des différent emplois du temps finaux.

- Vous pouvez retrouver la constitution des emplois du temps dans le fichier json/out/res.json, et le nombre de contraintes non respectées selon leur importance dans json/out/penalities.json.

- Vous pouvez visualiser ces emplois du temps dans votre navigateur web. Pour ce faire, executez la commande php -S localhost:8000, puis, sur votre navigateur, allez à ce lien : http://localhost:8000/index.html.

# 3 Fonctionnement de l'algorithme

Voici, dans de grandes lignes, le fonctionnement des différentes sections de notre algorithme évolutionnaire.

## 3.1 Avant tout

La première chose que nous faisons, c'est stocker les paramètres présents dans json/in/data.json dans des tableaux de classes. Cela permet de faciliter l'accès aux données, qui seront exploitées par la suite par notre algorithme.

## 3.2 Génome

Notre génome est constitué de deux tableaux d'entiers. Le premier, courses, est une abstraction des emplois du temps pour l'université. Dans notre optimisation, nous avons décidé de ne pas générer un emploi du temps par classe, car certaines salles sont limitées en nombre d'élèves. Nous avons donc décidé de faire des groupes au sein des classes, et nous créons donc un emploi du temps par groupe. Plusieurs groupes d'une même classe (uniquement d'une même classe) peuvent avoir cours ensemble si la salle le permet). Le tableau courses contient donc, pour chaque créneau de la semaine, pour chaque groupe, un entier indiquant le prof, et un entier indiquant la salle. Si l'entier correspondant au professeur vaut -1, alors cela signifie qu'il s'agit d'un temps libre. Le deuxième tableau, penalities, dénombre le nombre de contraintes non respectées selon leur importance.

## 3.3 Initialisation

Comme dans tout algorithme évolutionnaire, nous initialisons notre population de manière complètement aléatoire. Cela signifie que le tableau courses prend des valeurs aléatoires mais cohérentes : l'entier doit bien correspondre à une salle ou à un professeur (ou à un temps libre).

## 3.4 Croisement

Le croisement entre deux individus se fait de la manière suivante : pour chaque groupe, on choisi aléatoirement un créneau. L'enfant sera constitué de l'emploi du temps du premier parent jusqu'à ce créneau, et de l'emploi du temps du deuxième parent à partir de ce créneau.

## 3.5 Mutation

Nous affectons, pour chaque créneau, avec une faible probabilité, une classe et un professeur (ou un temps libre) de manière aléatoire à notre individu. Comme l'initialisation, ces entiers aléatoires doivent être cohérents.

## 3.6 Contraintes

Nous avons implémenté toutes les contraintes exigées par le projet, y compris les contraintes spécifiques aux salles et aux professeurs.

## 3.7 Évaluation

Ici nous analysons la qualité de notre individu. Nous calculons le "score" de l'individu en additionnant, à chaque fois que nous croisons une contrainte non respectée, son poids représentant son importance. Plus le score est faible, plus l'individu est répond au problème. Nous actualisons aussi le tableau penalities, dénombrant le nombre de contraintes non respectées selon leur importance. Nous avons décidé d'ajouter le poid de contrainte "Priority", qui permet, de donner un ordre de priorité dans les contraintes "Very High". Nous avons en effet remarqué que la création d'emploi du temps comportant le bon volume horaire pour chaque matière pour chaque classe est très difficile, rendre cette contrainte prioritaire est nécessaire pour obtenir des résultats exploitables. Sans "Priority", ces contraintes ne sont jamais respectées.

### 3.8 Après tout

Le meilleur emploi du temps trouvé est affiché sous forme de texte dans le terminal, et est stocké dans le fichier json/out/res.json. Le nombre de contraintes non respectées selon leur importance se trouve dans le fichier json/out/penalities.json.

## 4 Résultats

Nous avons tester notre algorithme évolutionnaire avec les paramètres d'universtité exigés par le projet. Nous avons implémenté toutes les contraintes

### 4.1 Ajout d'une salle de TP de Physique

Nous avons apporté une modification aux paramètres d'université, sinon le problème aurait été insoluble. Nous avons décidé d'ajouter une salle de TP de Physique (nous avons choisi arbitrairement 20 places maximum pour être cohérent avec la salle de TP d'Informatique), sinon les élèves n'ont pas de salle pour effectuer de type de TP.

### 4.2 Hyperparamètres

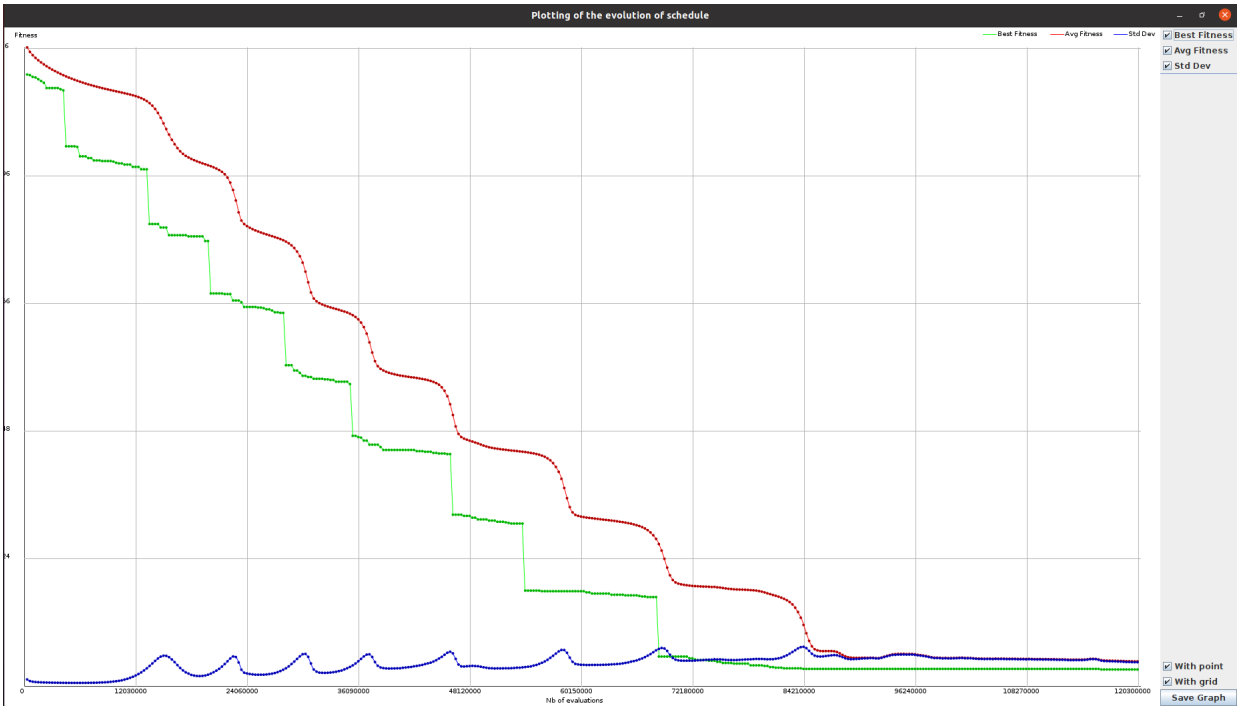
Nous avons choisi de prendre la plus grande population possible (300000 individus), car, l'algorithme convergenant très rapidement, notre objectif prioritaire était l'exploration. En suivant cette même philosophie, nous avons décidé de maintenir un taux de mutation relativement élevé (d'après Koza, ce taux doit être à 0.01, nous avons choisi 0.5), afin d'éviter une convergence prématurée. Nous avons choisi de réaliser 400 générations car nous estimions, de manière empirique, qu'il s'agit d'un nombre pertinent d'itération pour s'assurer de s'arrêter quelque peu de temps après que l'algorithme est complètement convergé (écart-type nul).

### 4.3 Temps d'exécution

Au vu des hyperparamètres choisis, une exécution dure environ 30 minutes.

## 4.4 Meilleur run

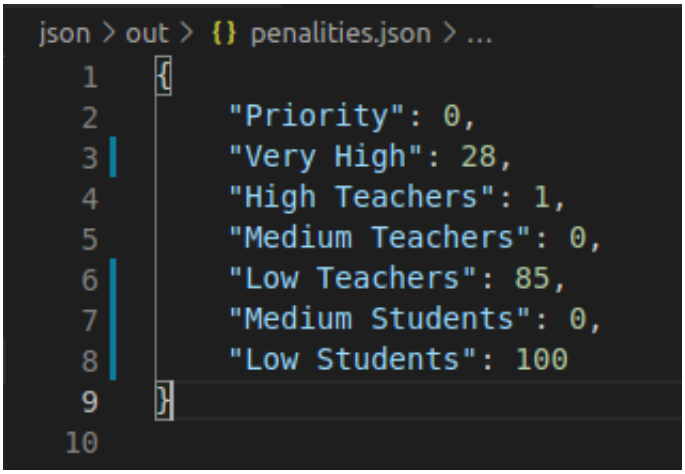
Voici l'évolution de notre meilleur run.




La seed permettant ce run était : 1611671840.

Les variations brutales de fitness correspondent au respect des contraintes "Priority".

Voici la liste des contraintes non respectées du meilleur emploi du temps que nous avons obtenu.

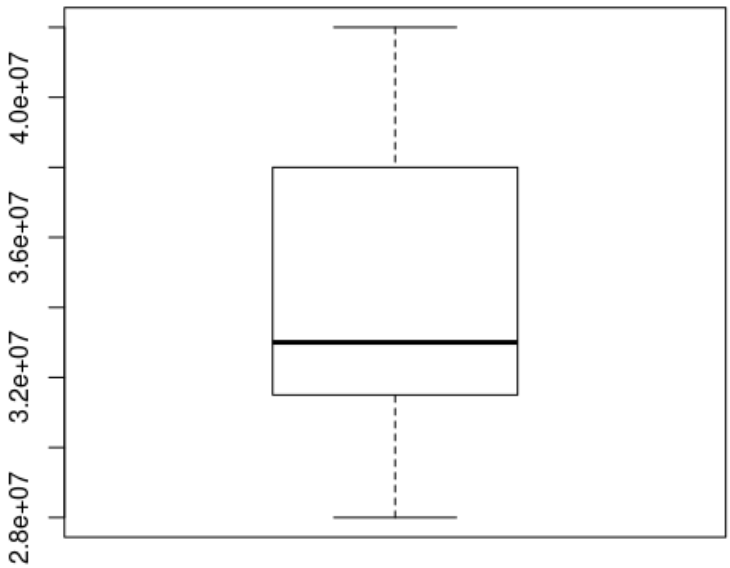



Toutes les contraintes prioritaires sont respectées, mais il reste 28 contraintes "Very High" non respectées.

Le score final du meilleur run est donc : 2.80004e+07.

## 4.5 Synthèse sur 20 runs

Voici une boite à moustache résumant les valeurs de fitness obtenus sur 20 runs.




### 4.6 Interprétation

Nous remarquons que quand nous lançons notre algorithme, nous parvenons à ne plus avoir aucune contrainte "Priority". Cependant, nous obtenons des emplois du temps ne respectant pas en moyenne entre 30 et 40 contraintes "Very High". Cela signifie que nous avons trouvé aucun emploi du temps répondant vraiment au problème. Nous sommes tout de même satisfait, au vu de la complexité de ce problème, d'avoir pu réduire autant le nombre de contraintes "Very High" (sachant que nous les avons toutes implémentées, même celles qui sont spécifiques).

## 5 Améliorations possibles

Nous sommes convaincu que pour améliorer nos performances, nous devons continuer à explorer davantage. Un nombre plus élevé d'individus aurait pu être une solution. Nous avons aussi remarqué, lors de l'évolution de notre algorithme, qu'une fois que les contraintes "Priority" sont passées,

l'optimisation devient toute autre, et des hyperparamètres différents aurait été plus pertinent. On aurait pu donc imaginer une résolution en deux phases, avec un changement des hyperparamètres au moment ou les contraintes "Priority" sont globalement respectées dans la population. Une dernière hypothèse d'amélioration : les croisements. Il s'agit d'une phase de l'algorithme que nous pouvons renforcé, et qui pourrait exploiter un peu plus les liens entre les deux parents afin d'obtenir un effet d'hétérosis.

# 6 Conclusion

Tout d'abord, ce projet nous a fait comprendre conscience d'un point essentiel : faire des emplois du temps, c'est pas facile. Nous sommes satisfait d'avoir implémenter un algorithme minimisant toutes les contraintes "Priority" et un nombre très important de contraintes "Very High". Nous pensons, au travers des améliorations que nous proposons, qu'il aurait sûrement été possible d'obtenir des emplois du temps répondant vraiment au problème.