TPP Master 1: Élaboration d'un horaire d'examen grâce à un algorithme génétique

Auteurs:

RACHIK Nesrine
ADARDOUR Rida

Responsable:

MORIAME Martin
TILMAN Eve
SIMAL Cédric
DORCHAIN Marie

Décembre 2023

Contents

1 Introduction

L'objectif de ce travail est d'élaborer un emploi du temps d'examens grâce à un algorithme génétique. Les algorithmes génétiques font partie de la famille des algorithmes évolutionnistes, visant à obtenir une solution approchée à un problème d'optimisation.

Le cahier des charges vise à concevoir un programme permettant de générer un emploi du temps d'examens incluant tous les cours requis, tout en minimisant les conflits horaires.

Afin de concrétiser ce cahier des charges, nous avons utilisé le langage de programmation Python.

2 Élaboration d'un horaire d'examen grâce à un algorithme génétique

2.1 Les classes définies

On a défini quatre classes :

2.1.1 Classe Teacher

Cette classe possède trois attributs, à savoir le nom de l'enseignant, la liste des cours dont il est responsable, ainsi qu'un DataFrame contenant ses disponibilités en fonction des jours (indices des colonnes) et des horaires (indices des lignes).

```
teacher1.name
```

```
'Bierwart Francois-Gregoire'
```

```
teacher1.courses
```

```
['Algèbre Supérieur', 'Analyse complexe', 'Mécanique des fluides']
```

```
teacher1.avl
```

	day1	day2	day3	day4	day5	day6	day7	day8	day9	day10
morning	1	0	0	0	0	0	0	0	1	1
noon	1	1	0	0	1	1	0	0	1	0

FIG1: Les attributs de la classe Teacher

2.1.2 Classe Student

Cette classe possède deux attributs à savoir le nom de l'étudiant ainsi qu'une liste contenant les cours qu'il suit.

```
student2.name
```

```
'Elsa Durand'
```

```
student2.courses
```

```
['Electricité',
 'Philosophie des sciences',
 'Soin aux créatures magiques',
 'Defense contre les forces du mal',
 'Botanique',
 'Algorithmique',
 'Analyse Réelle',
 'Anglais',
 'Algèbre linéaire',
 'Vol sur balais',
 'Mécanique des fluides']
```

FIG2: Les attributs de la classe Student

2.1.3 Classe Schedule

Il s'agit du chromosome de notre algorithme génétique. Cette classe possède deux attributs, à savoir la variable fitness qui est égale à la fonction objectif qui évalue notre calendrier d'examen ainsi qu'un DataFrame qui représente les horaires d'examens en fonction des jours et des horaires.

```
sched1.fitness
```

```
2711
```

```
sched1.sc
```

	morning	noon
day1	[Théorie qualitative des systèmes dynamiques, ...	[Mécanique des fluides, Philosophie des scienc...
day2	[Analyse complexe, Programmation]	[Astronomie, IDM, Histoire de la magie]
day3	[Anglais, Defense contre les forces du mal]	[]
day4	[Soin aux créatures magiques, Mécanique du poi...	[Sortileges]
day5	[Sciences religieuses, Analyse Réelle]	[Analyse numérique]
day6	[Botanique]	[]
day7	[Mesure et intégration]	[Probabilités]
day8	[Algèbre Supérieur, Divination, FCO]	[]
day9	[]	[Vol sur balais]
day10	[]	[Physique, Electricité]

FIG3: Les attributs de la classe Schedule

2.1.4 Classe gene

Il s'agit du gène de notre algorithme génétique. Cette classe possède trois attributs à savoir le nom du cours de l'examen, son jour ainsi que son horaire.

```
genex.exam
```

```
'Théorie qualitative des systèmes dynamiques'
```

```
genex.time
```

```
'noon'
```

```
genex.day
```

```
'day1'
```

FIG4: Les attributs de la classe gene

2.2 Stockage et structure des données

- Chargement des données:
  Pour le chargement et la sauvegarde des informations des enseignants et des étudiants, nous avons défini une fonction appelée *charge_data*() qui facilite le chargement des données de tous les professeurs et étudiants à partir des dossiers "Profs" et "Students". Ces données sont ensuite transformées en deux listes distinctes : "*teacher_list*" et "*student_list*". La liste "*teacher_list*" contient des instances de la classe Teacher, tandis que la liste "*student_list*" contient des instances de la classe Student.

- Structures de données:
  Pour accomplir cette tâche, notre fonction "*charge_data*()" calcule automatiquement le nombre de jours et crée une liste appelée "*Exam_days*", qui répertorie tous les jours en leur attribuant un nom, tel que "*day*1" pour le premier jour. En ce qui concerne les horaires, nous considérons deux possibilités : "morning" ou "noon", que nous regroupons dans une liste nommée "times". Pour stocker la disponibilité des enseignants, nous utilisons la structure de données DataFrame, plaçant les jours en colonnes et les horaires en lignes. Lorsque le professeur est disponible, nous attribuons la valeur 1, sinon nous utilisons la valeur 0.

	day1	day2	day3	day4	day5	day6	day7	day8	day9	day10
morning	1	0	0	1	0	0	1	0	1	1
noon	1	1	0	0	0	0	0	1	0	0

FIG1: La disponibilité d'un prof en DataFrame

- Chargement des cours:
  La fonction *course_charge*() permet de charger tous les cours des examens dans une liste appelée 'Cours'.

```
courses = charge_courses(teacher_list)
courses
```

```
['Defense contre les forces du mal',
 'Sortileges',
 'Botanique',
 'Divination',
 'Soin aux creatures magiques',
 'Astronomie',
 'Potions',
 'Histoire de la magie',
 'Metamorphose',
 'Vol sur balais']
```

FIG4: Liste des cours

- Chargement de la liste des étudiants par cours:
  La fonction *charge_dfstudent*() retourne une Dataframe "*df_student*" qui permet d'identifier tous les étudiants suivant un cours spécifique.

```
df_student["Botanique"]
```

```
Student    [Adardour Rida, Boudon Pauline, Fiabge Kolman,...
Name: Botanique, dtype: object
```

FIG4: Identification des étudiants par cours

- Identification des enseignants responsables de chaque cours:
  La fonction *charge_dfteacher*() retourne une Dataframe "*df_teacher*" permet de déterminer l'enseignant responsable de chaque cours.

```
df_teacher["Botanique"]
```

```
Teacher    Name: Moriame Martin
Name: Botanique, dtype: object
```

FIG4: Professeur responsable d'un cours

2.3 Fonction objectif

Pour implémenter notre fonction objectif, nous avons commencé par classifier les contraintes imposées sur les emplois du temps. Dans notre cas, en se basant sur l'énoncé du projet, nous avons choisi de considérer trois contraintes auxquelles nous avons assigné des coefficients de pondération. Ces contraintes sont classées en deux catégories : deux contraintes fortes de coefficient α et une contrainte faible de coefficient β tel que $\alpha > \beta > 0$. Pour éviter la violation des contraintes fortes et orienter notre algorithme et notre recherche vers une solution faisable et autorisée, on va choisir un très grand coefficient de pondération α par rapport à celui de la faible contrainte.

- Première contrainte forte (Enseignants):
  Il s'agit d'une contrainte selon laquelle un enseignant ne peut pas superviser plus d'un examen simultanément. Pour calculer le nombre de contraintes de ce type dans un emploi du temps, nous avons défini une fonction "*calculate_high_constraints*1" qui évalue le nombre de conflits de ce genre.

- Deuxième contrainte forte (Enseignants):
  Cette contrainte stipule qu'un enseignant ne peut pas avoir un examen à surveiller à un horaire qui ne correspond pas à ses disponibilités. Pour quantifier le nombre de contraintes de ce type dans un emploi du temps, nous avons créé une fonction "*calculate_high_constraints*2" qui permet de calculer le nombre de conflits de ce type.

- Contrainte faible (Étudiants):
  Cette contrainte stipule qu'un étudiant ne peut pas avoir plus d'un examen programmé au même horaire. Pour évaluer le nombre de contraintes de ce type dans un emploi du temps, nous avons élaboré une fonction "*calculate_low_constraints*"qu
  permet de déterminer le nombre de conflits de ce type.

Dés qu'on aura fini d'implémenter ces contraintes, on va calculer le fitness d'un individu en multipliant le nombre de contraintes de chaque catégorie par son coefficient de pondération et on fait la somme pour trouver le fitness d'un individu Schedule(emploi du temps).

```
fitness = 100*(high_constraints1 + high_constraints2) + low_constraints
schedule.fitness = fitness
```

fig1: Évaluation d'un individu(emploi du temps)

3 Implémentation des opérateurs génétiques

3.1 Initialisation aléatoire d'une population

Pour générer un emploi du temps aléatoire, nous avons implémenté la fonction "*random_schedule*", qui crée un emploi du temps en parcourant tous les cours dispensés par chaque professeur. Elle attribue de manière aléatoire à chaque cours un horaire et un jour, puis place ces cours dans une cellule du DataFrame qui constituera notre emploi du temps aléatoire. Cette sélection aléatoire nous offre la possibilité de découvrir de nouvelles solutions.

	morning	noon
day1	[]	[Théorie qualitative des systèmes dynamiques]
day2	[Potions, Anglais]	[Analyse complexe, Botanique, Statistiques, Vo...
day3	[Philosophie des sciences, Astronomie, Analyse...	[]
day4	[Mécanique des fluides, Sciences religieuses, ...	[Mesure et intégration]
day5	[Algèbre Supérieur, FCO]	[IDM]
day6	[Divination]	[Programmation]
day7	[Mécanique du point]	[Algèbre linéaire]
day8	[Algorithmique]	[Defense contre les forces du mal]
day9	[Sortileges, Metamorphose]	[]
day10	[Analyse Réelle, Probabilités, Histoire de la ...	[Soin aux créatures magiques]

FIG1: Un emploi du temps aléatoire

Après avoir défini une fonction capable de générer un emploi du temps aléatoire, nous allons maintenant définir une autre fonction *initialize_population*, qui permet de créer une liste contenant les individus de notre population aléatoire générés par la fonction *random_schedule*.

```
population1 = initialize_population(10,teacher_list)
```

Initialisation d'une population de 10 éléments

```
population
```

```
[<__main__.Schedule at 0x241645ee960>,
 <__main__.Schedule at 0x241645e89b0>,
 <__main__.Schedule at 0x241645ea810>,
 <__main__.Schedule at 0x241645e9310>,
 <__main__.Schedule at 0x241645e96d0>,
 <__main__.Schedule at 0x241645e80b0>,
 <__main__.Schedule at 0x241645e7b60>,
 <__main__.Schedule at 0x241645e5eb0>,
 <__main__.Schedule at 0x241645e63f0>,
 <__main__.Schedule at 0x241645e63c0>]
```

Liste des chromosomes/Schedule d'une population de 10 éléments

3.2 Sélection pour la reproduction

Dès que notre population est initialisée, nous passerons à la deuxième étape qui consiste à sélectionner les meilleurs parents. Pour cela, nous utiliserons la méthode de sélection par roulette, qui attribue un secteur d'une roue à chaque individu. Ce secteur est proportionnel à son adaptation (fitness). Nous faisons tourner la roue et lorsque celle-ci cesse de tourner, nous sélectionnons l'individu correspondant au secteur désigné par un curseur pointant sur un secteur particulier après l'arrêt de la rotation.

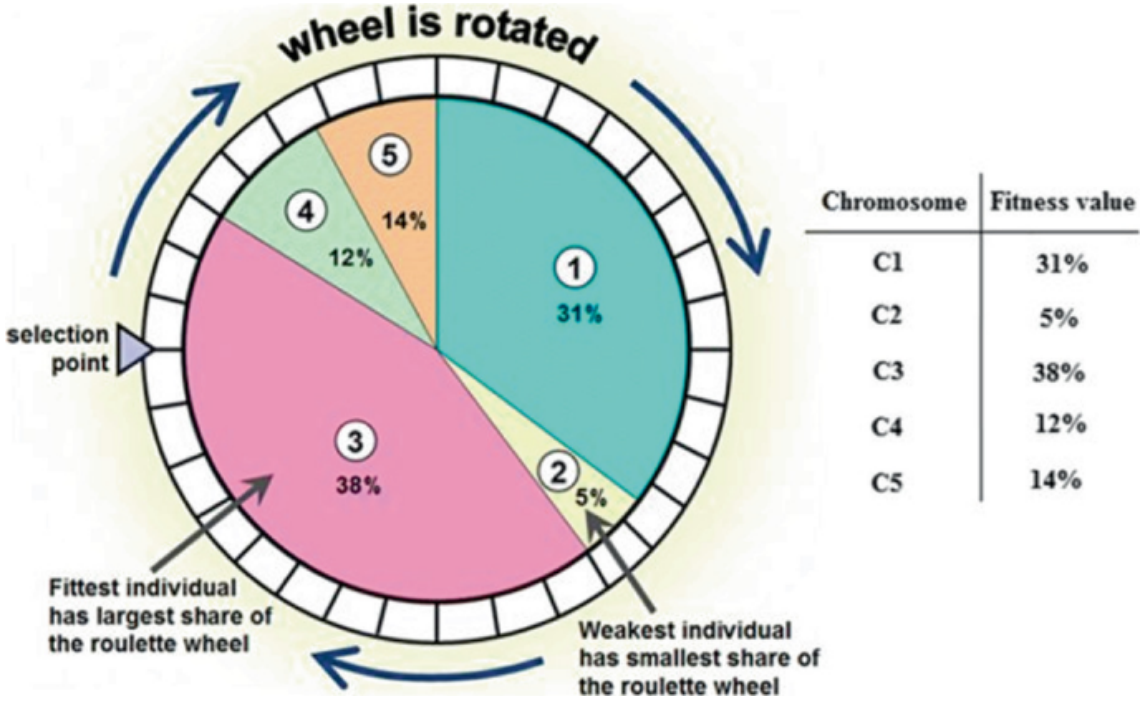


FIG1: La méthode de sélection par roulette

3.3 Croisement

3.3.1 Le croisement uniforme

La méthode de croisement qui a été utilisée est le croisement uniforme. Le principe est le suivant : on prend deux parents, A et B ; pour chaque gène, on lance une pièce de monnaie ; si c'est face, l'enfant prend le gène du parent A ; si c'est pile, l'enfant prend le gène du parent B.

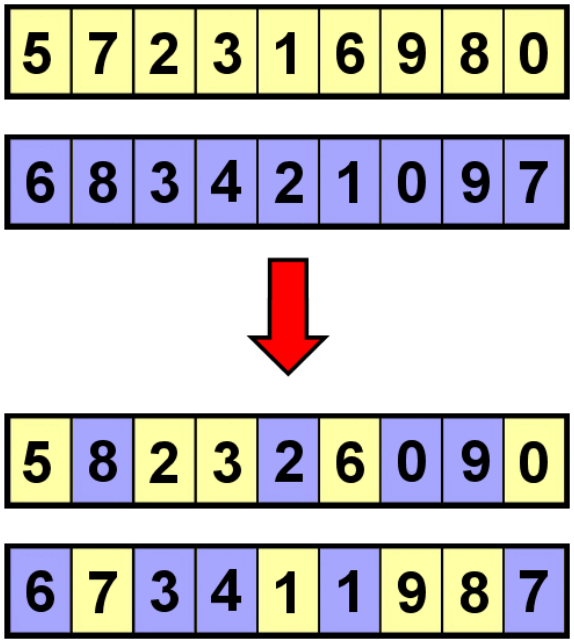


FIG: Le croisement uniforme

3.3.2 Pourquoi ce choix ?

Tandis que le croisement à un point et le croisement à deux points convergent rapidement vers une solution, le croisement uniforme donne généralement des résultats supérieurs à ceux produits par les deux autres méthodes.

3.3.3 Fonctionnement

- **Première étape: Extraction des gènes d'un chromosome**

  Cette première étape permet à partir d'un emploi du temps ou un chromosome d'extraire les gènes dont il est composé. Cela se fait en utilisant la

fonction *generate_genes* qui prend en argument notre emploi du temps et retourne une liste qui contient des instances de la classe gène.

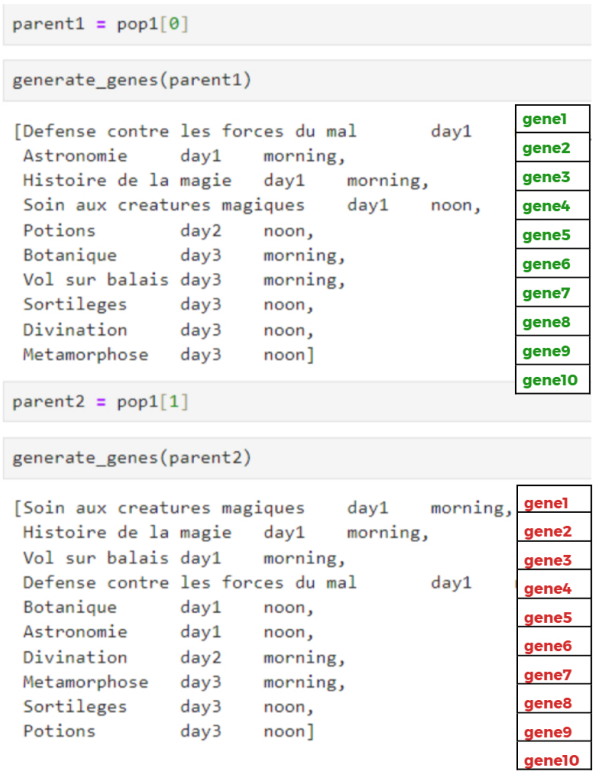


FIG: Extraction de gènes d'un individu

- **Deuxième étape: Croisement uniforme entre deux parents**

  Après l'extraction des gènes de nos parents, nous entamons la phase cruciale de notre algorithme, à savoir le croisement. Nous parcourons la liste de tous les cours et avec une probabilité de 0.5, nous effectuons un échange de gènes correspondant à chaque cours. Cette méthode de sélection de cours nous permet d'éviter les répétitions dans la liste de gènes des deux enfants, assurant ainsi un résultat valide. La figure suivante illustre cet échange de gènes entre deux parents. Comme le montre clairement la figure,

FIG: croisement uniforme: échange de gènes entre deux parents

notre croisement produit deux enfants valides: **s**ans répétition de cours, et chacun contenant exactement 10 cours. On remarque bien dans la figure que notre croisement produit deux enfants valides: **il y a pas de répétition de cours** et contiennent exactement 10 cours

- **Troisième étape: regroupement des gènes pour créer un individu** Après le croisement, nous disposons désormais de deux listes de gènes distinctes. Il nous incombe maintenant de regrouper ces gènes afin de créer les deux enfants (emplois du temps). Il convient de rappeler que ces deux listes

renferment des instances de la classe "gene", chaque instance comportant trois attributs : "exam", "time" et "day". À partir de ces attributs, nous pouvons positionner chaque gène dans l'emplacement approprié de l'emploi du temps. Par exemple, le "gene4" du "child1" (en vert) a pour attributs "day1" et "noon" pour le jour et l'heure, respectivement. Ainsi, nous le plaçons dans la case de coordonnées (day1, noon), en suivant la même démarche pour le reste des gènes. À cette fin, nous avons défini une fonction, *genes_to_schedule*, qui transforme une liste de gènes en un emploi du temps valide.

La figure suivante montre le regroupement de gènes des parents dans leurs emplois du temps, en respectant les emplacements indiquées par les attributs des gènes day et time:

Les Parents 1 et 2 avant le croisement uniforme

parent1	morning	noon
day1	[gene1,gene2,gene3]	[gene4]
day2		[gene5]
day3	[gene6, gene7]	[gene8, gene9, gene10]

parent2	morning	noon
day1	[gene1,gene2,gene3]	[gene4, gene5, gene6]
day2	[gene7]	
day3	[gene8]	[gene9, gene10]

FIG1: Avant le croisement uniforme

Dans la figure suivante, nous observons les deux enfants créés après le regroupement des listes de gènes: l'enfant 1 a hérité des gènes 3, 5, 7, 8 et 10 du parent 2, tandis que l'enfant 2 a pris les gènes 5, 6, 7, 9 et 10 du parent 1. En examinant l'attribut "exam" de chaque gène, nous constatons que notre emploi du temps ne présente aucune répétition d'examens et contient exactement 10 cours: **C'est un cours valide**

Les Enfants 1 et 2 après le croisement uniforme

enfant1	morning	noon
day1	[gene1,gene2,gene3, gene3]	[gene4, gene5]
day2	[gene7]	
day3	[gene8]	[gene8,gene10]

enfant2	morning	noon
day1	[gene2,gene1]	[gene4, gene6]
day2		[gene5]
day3	[gene6,gene7]	[gene10,gene9,gene9]

FIG2: Après le croisement uniforme

3.4 Mutation

Pour introduire la diversité des chromosomes dans les populations, on va utiliser la mutation. La mutation a l'avantage d'éviter les minima locaux et empêcher la forte ressemblance entre les chromosomes de la population.

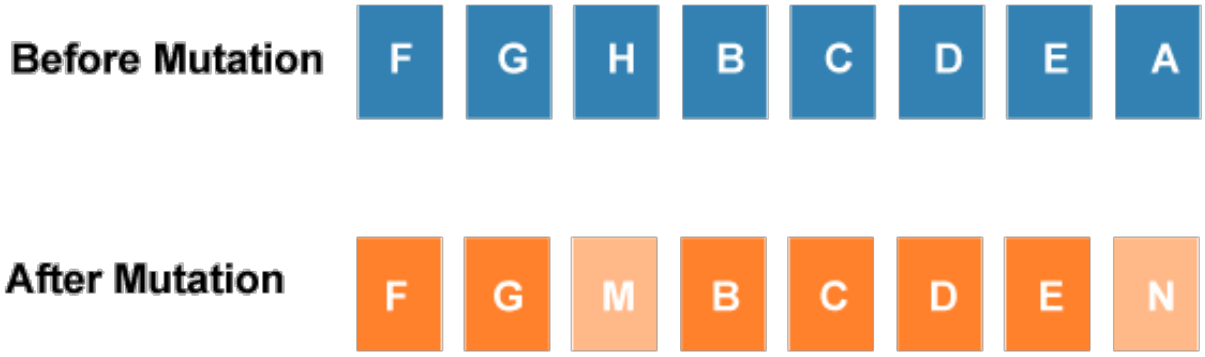


FIG2: La mutation

Dans notre cas, on va implémenter l'opérateur génétique en suivant ces étapes:

- On choisit le chromosome sur lequel on veut effectuer la mutation.
- On extrait sa liste de gènes ou de cours.
- Pour éviter d'altérer les meilleurs solutions, on va choisir aléatoirement que le tiers ou le quart des gènes dont le chromosome est composé, et on va leur assigner aléatoirement un nouveau jour et une nouvelle heure.
- Enfin, on regroupe les nouveaux gènes de notre chromosome muté, pour créer un emploi du temps.

  Remarque: La mutation sera appliquée à chaque génération mais avec une faible probabilité.

La figure suivante illustre les résultats d'une mutation:

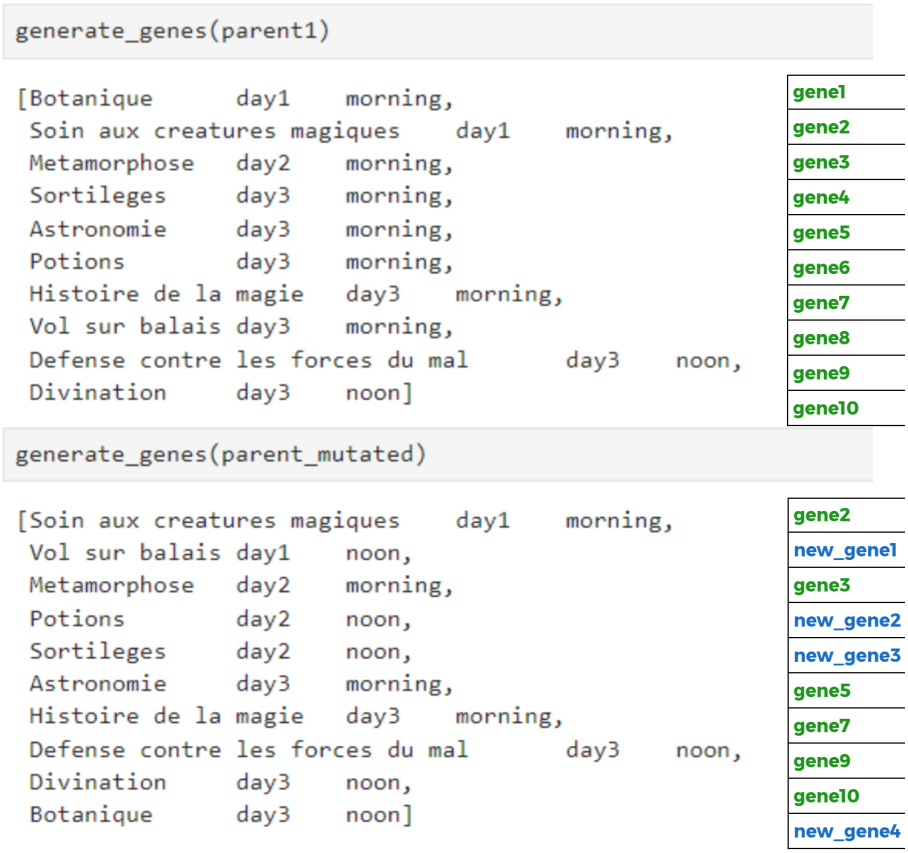
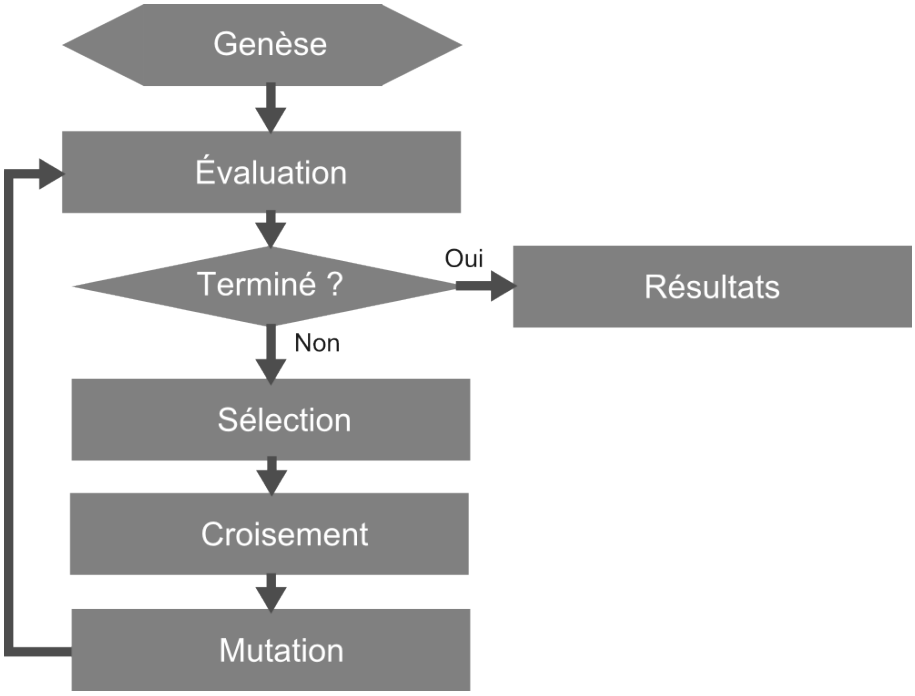


FIG2: La mutation

On remarque que le parent muté contient quatre nouveaux gènes(en Bleu) qui ont été crées par la mutation.

4 Fonctionnement de notre algorithme



Étapes de l'algorithme

- **Étape 1: Initialisation**

  Créer aléatoirement une population de chromosomes de taille N. Chaque chromosome est représenté par un objet de la classe Schedule qui a deux attributs le fitness et un DataFrame qui constitue notre emploi de temps sous forme de tableau.

- **Étape 2: Évaluation**

  On évalue chaque chromosome de la population en se basant sur les ,contraintes qu'on a défini précédemment. On associe à chaque objet de la classe Schedule un fitness/score qui correspond au nombre de conflits.

- **Étape 3: Sélection**

  La sélection consiste à choisir N chromosomes les mieux adaptés en utilisant la technique de la roulette. Cela nous permet d'avoir une population de solution la plus proche de converger vers l'optimum global

- **Étape 4: Croisement**

Nous allons désormais créer une nouvelle population en utilisant le croisement uniforme. Ce processus consiste à choisir deux parents chromosomes parmi la population sélectionnée, fusionner leurs gènes pour produire deux enfants chromosomes, et s'arrêter une fois que la nouvelle population atteint la taille N.

- **Étape 5: Mutation**

  La population issue du croisement passe à l'étape suivante, qui est la mutation. Nous sélectionnons, selon une probabilité faible, des chromosomes que nous mutons en modifiant l'emplacement et les attributs (jour d'examen, heure d'examen) de leurs gènes. Nous réévaluons ensuite la nouvelle population obtenue.

- **Dernière étape: Condition d'arrêt**

  Nous extrayons le meilleur chromosome de notre population, qui représente un maximum local. Si ce chromosome satisfait toutes les contraintes, l'algorithme s'arrête et nous affichons le résultat. Si les contraintes ne sont pas satisfaites, l'algorithme se réitère jusqu'à atteindre le nombre maximal de générations fixé au début.

5 Résultats de notre algorithme

Après avoir terminé l'implémentation de notre algorithme génétique, on va effectuer maintenant des tests sur notre code avec le petit jeu de données "tests Projet Math-App", ensuite on va le mettre à l'épreuve sur un ensemble de données volumineux "Data Cluster Projet Math-app" en utilisant le cluster. Cela nous permettra d'exécuter notre code avec différentes combinaisons de paramètres.

5.1 Application sur les données "tests Projet Math-App"

5.1.1 Meilleur résultat

Pour tester notre code et la validité des solutions, on a lancé notre algorithme sur les données "tests Projet Math-App" et on a trouvé plusieurs solutions. On va exposer la meilleure solution ainsi que les paramètres qui ont été utilisés.

Au niveau des paramètres on a utilisé les paramètres suivants:

- Pénalité Profs = 10 (On a augmenté la pénalité des profs pour garantir la satisfaction des fortes contraintes)
- Pénalité étudiants = 1
- Taille de la population = 50
- Nombre de générations = 100
- Taux de mutation = 0.4 (mutation élevée pour maintenir la diversité dans chaque génération)
- Taux de croisement = 1 (Pour assurer la reproduction et la création des enfants meilleurs que leurs parents)

Le résultat suivant est le meilleur parmi toutes les autres solutions trouvées:

```
99 Le fitness de la solution initiale: 55       le fitness de la nouelle solution:  11
                                                morning                                 noon  fitness
day1  [Potions, Soin aux creatures magiques, Vol sur...                          [Divination]       0
day2                [Defense contre les forces du mal]  [Histoire de la magie, Sortileges]       0
day3                                       [Astronomie]          [Botanique, Metamorphose]       0
PS C:\Users\ridaa\Downloads\TPP>
```

Meilleur résultat trouvé

- fitness = 11
- Il y a pas de conflit de type contraintes fortes (Regarder la colonne droite de la Dataframe nommée fitness qui compte le fitness des fortes contraintes)
- 11 conflits au niveau des faibles contraintes.

On a vérifié ce résultat à la main ainsi que la validité de la solution. C'est un résultat valide et correct.

5.2 Application sur les données "Data Cluster Projet Math-app"

5.2.1 Analyse des résultats avec le Cluster

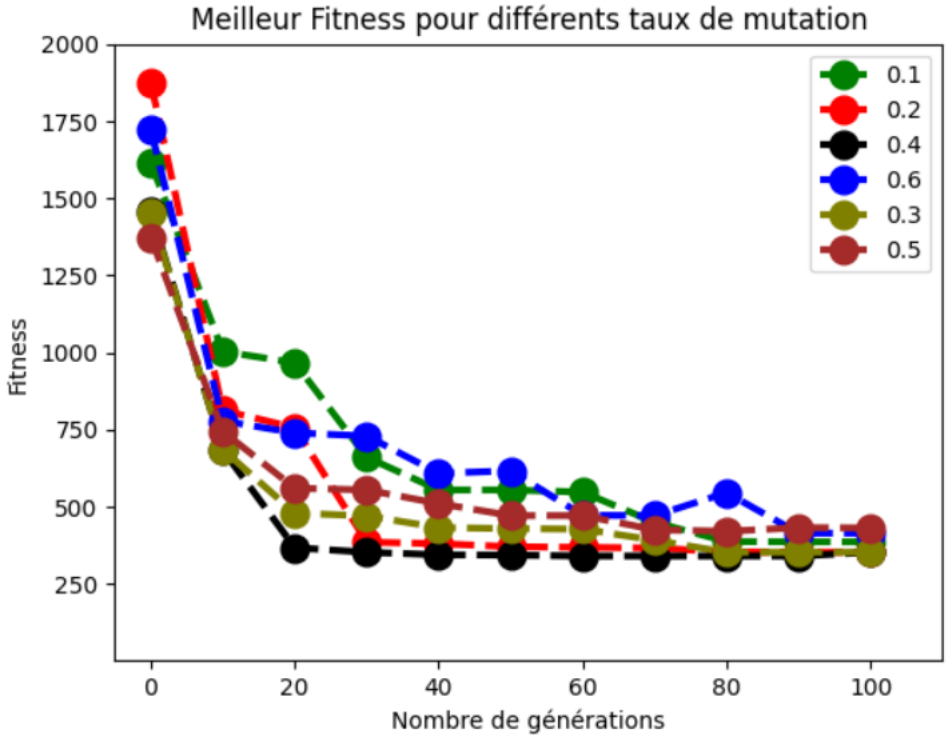
Sur cet ensemble de données volumineux, nous allons tester notre code avec différentes combinaisons de paramètres. À la fin, nous récupérerons nos résultats, les comparerons, les interpréterons, et afficherons la meilleure solution trouvée.

Après avoir testé notre code, sur 50 combinaisons de paramètres,nous avons obtenu plusieurs résultats différents, on va se contenter de montrer que les meilleurs résultats. :

Pour la combinaison de paramètres suivante:

- Pénalité Profs = 100
- Pénalité étudiants = 1
- Taille de la population = 50
- Nombre de générations = 100
- Taux de mutation = 0.4
- Taux de croisement = 1

On a tracé ce graphique qui illustre l'influence du taux de mutation et de la mutation en général sur les résultats finaux.



Résultats en fonction du taux de mutation

Dans ce graphe, on observe que plus la valeur de la mutation augmente, plus la qualité de la solution diminue et la convergence est plus lente. De même, lorsque la mutation est proche de 0 ou de 0.1, la solution n'est pas optimale, et la convergence est également lente.

En revanche, lorsque la mutation est correctement réglée à des valeurs telles que 0.2, 0.3 et 0.4, la convergence est rapide, et la solution atteint un niveau optimal.

Ce résultat peut s'expliquer par le fait qu'une mutation importante introduit de la diversité, mais peut également altérer les meilleures solutions. D'un autre côté, une mutation faible conserve les bonnes solutions, mais ne contribue pas à apporter de la diversité dans la population..

5.2.2 Meilleure solution

Après avoir comparé tous nos résultats on a trouvé cette meilleure solution pour les mêmes paramètres:

```
99 Le fitness de la solution initiale: 1457      le fitness de la nouelle solution:  340
                                                 morning                                    noon  fitness
day1                                               [FCO]                      [Potions, Anglais]        0
day2                                [Sciences religieuses]                        [Analyse Réelle]        0
day3                                      [Vol sur balais]                [Algorithmique, Physique]        0
day4                                [Botanique, Sortileges]            [Soin aux créatures magiques]        0
day5                   [Defense contre les forces du mal]                  [Mécanique des fluides]        0
day6                    [Programmation, Algèbre linéaire]                [Divination, Electricité]        0
day7   [Théorie qualitative des systèmes dynamiques, ...  [Metamorphose, Philosophie des sciences]        0
day8                                [Histoire de la magie]         [Mécanique du point, Astronomie]        0
day9                     [Statistiques, Algèbre Supérieur]                                  [IDM]        0
day10              [Analyse numérique, Analyse complexe]                          [Probabilités]        0
radardou@gauss26:~/Documents/TPP$
```

Meilleur résultat trouvé

Satisfaction des contraintes:

- Fortes contraintes sur les Profs: Toutes les fortes contraintes ont été satisfaites. Dans le tableau à droite, la colonne "fitness" affiche le nombre de conflits pour chaque jour, montrant qu'il n'y a aucun conflit de type fortes contraintes, avec un total de 0 conflits.
- Faibles contraintes: Les faibles contraintes concernant les étudiants ne sont pas toutes satisfaites, et nous avons enregistré un total de 340 conflits.

Le score de fitness de ce résultat est de 340, ce qui représente le meilleur résultat obtenu parmi toutes les itérations que nous avons effectuées.

6 Conclusion

Conclusion Au cours de notre projet, nous avons tiré des conclusions et noté des observations cruciales sur le fonctionnement de cet algorithme. Nous en énumérons quelques-unes ci-dessous:

- La maintenance de la diversité joue un rôle très important sur la qualité de la solution
- Il est essentiel d'éviter une convergence prématurée qui pourrait se stabiliser dans un minimum local.
- Les opérateurs de croisement doivent exploiter les gènes des parents pour garantir l'efficacité de l'algorithme.
- Il est nécessaire d'éviter les mutations qui compromettent la validité de la solution, telles que la répétition d'un cours deux fois dans un même emploi du temps.
- Le croisement uniforme est très disruptif par rapport au croisement mono-point et multi-point.
- Il faut toujours veiller à ne pas croiser les meilleurs parents entre eux au risque d'un convergence prématurée.
- La pénalité des fortes contraintes doit être nettement supérieure à celle des faibles contraintes pour ne pas enfreindre les fortes contraintes.
- Une très grande population ne donne pas forcément une meilleure solution.