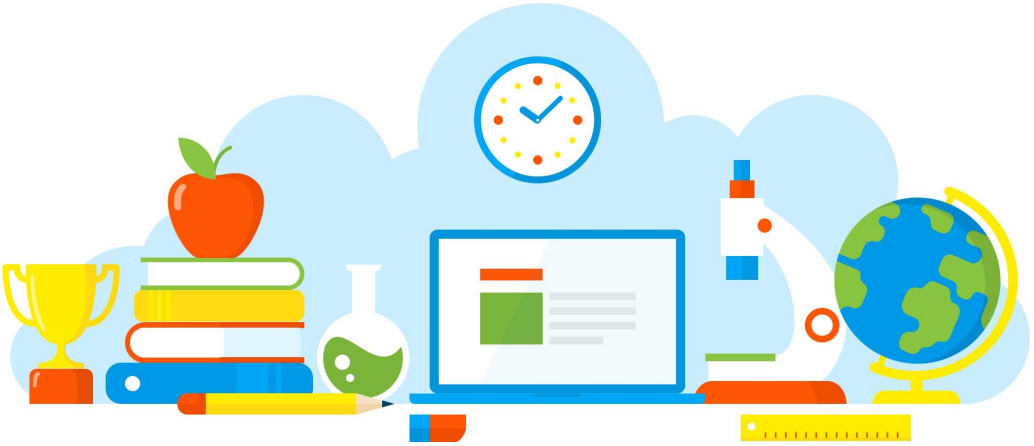


Rapport Projet de Métaheuristiques

Réalisé par:

Yassine TAHRI
Pierre FAJOL
FABIOLA FEYA

Professeurs

Sonia YASSA

Introduction

Nous sommes trois étudiants de CYTECH à avoir réalisé un projet de modélisation d'un problème multi-objectifs et l'implémentation de sa résolution grâce à une technique de métaheuristiques.

Ce rapport intervient dans le cadre du cours de Métaheuristiques dispensé pour notre dernière année d'études de formation ingénieur en intelligence artificielle que l'on réalise en alternance. Il comprend toutes les étapes qui nous ont permis de réaliser ce projet.

Nous avions le devoir de choisir un problème multi-objectifs parmi ceux régulièrement rencontrés dans le domaine des iot et du Cloud. On distingue par exemple des problèmes tels que : Workflow Scheduling, Task Offloading,...

En tant qu'ingénieur en informatique, nous pourrions être très certainement confrontés à ce genre de problème, et il est donc utile pour nous de connaître les techniques de résolution de celui-ci via les Métaheuristiques.

Nous vous présenterons dans ce rapport, le choix de notre problème multi-objectifs, ses variables de décision, ses fonctions objectives et ses contraintes. Nous justifierons nos choix de résolution et présenterons nos résultats.

Problème

Task Scheduling

Le cloud computing est une nouvelle technologie qui fournit des ressources virtuelles, évolutives et dynamiques aux utilisateurs sur la base d'un service de paiement à l'utilisation. Cette technologie dépend du réseau. En raison de l'échelle du réseau impliqué, certains services tels que les applications de commerce électronique utilisent tout le réseau. Le cloud computing se développe pratiquement en trois étapes : le calcul distribué, le calcul parallèle et le calcul en grille. Il peut être aussi classé en trois types : les clouds publics que chacun peut enregistrer et utiliser pour ses services, les clouds privés qui sont exploités sans limitation de bande passante réseau, de visibilité de la sécurité et de spécifications réglementaires au sein de l'organisation et les clouds hybrides qui fusionnent les clouds privés avec les ressources du cloud public.

Un aspect important du cloud computing est la planification des flux de travail qui est le processus de mappage des tâches dépendantes aux ressources disponibles en tenant compte des contraintes de qualité de service (QoS). La planification des flux de travail entraîne un coût de communication et de calcul important. C'est ce qu'on appelle un problème de Task Scheduling.

Le problème consiste à trouver la meilleure répartition de tâches données entre les serveurs (ressources) disponibles. Il est question de trouver la meilleure répartition tout en veillant à tirer le maximum de bénéfice en matière par exemple de consommation d'énergie, de temps d'exécution ,de coût d'exécution, de latence ou encore d'utilisation de ressources.

Le sujet de notre projet porte donc sur la thématique du task scheduling. On souhaite allouer un nombre de tâches définies sur différentes unités de calcul disponibles. Une tâche ici étant définie par un nombre d'instructions et un serveur par un nombre d'instructions par seconde et un coût unitaire d'exécution. L 'objectif est de répartir la charge sur ces différents serveurs de façon à **minimiser** le *temps d'exécution*, le *coût d'exécution* et *l'utilisation des ressources*.

Les données de ce projet sont les suivantes :

- **Les serveurs** se distinguent par leurs capacités de calculs exprimées en nombre d'instructions par seconde ainsi que leur coût d'utilisation, représenté en euros par seconde.

- **Les tâches** se différencient par leurs nombres d'instructions.

Modélisation mathématique

Ce problème met en scène 3 variables de décision :

Domaine	Métriques	Paramètres	Variable de décision	Fonctions objectifs
Task scheduling	***Temps d'exécution*** ***Coût d'exécution*** ***Ressources utilisées***	**S** : Nombre de serveurs **T** : Nombre de tâches **Ni** : Nombre d'instructions d'une tâche i **Cj**: Coût d'exécution unitaire d'un serveur j **Nsj**: Nombre d'instructions par seconde d'un serveur j	Ti = Temps d'exécution d'une tâche Cvmi = coût d'exécution d'une machine virtuelle R = nombres de ressources utilisées S = ensemble des solutions pareto	➔ F1= **Min (Max(Ti))** avec **Ti = Ni/Nsj** (j étant le serveur alloué pour l'exécution de la tâche i) ➔ F2 **= Min (Rs)** ➔ F3 **= Min** $\sum_{i=0}^{n} Ti * Cvmi$

Algorithme

NSGA-II

Il s'agit d'un problème multi-objectif avec 3 fonctions à minimiser. Pour résoudre ce problème, nous avons choisi d'utiliser l'algorithme **NSGA 2.**

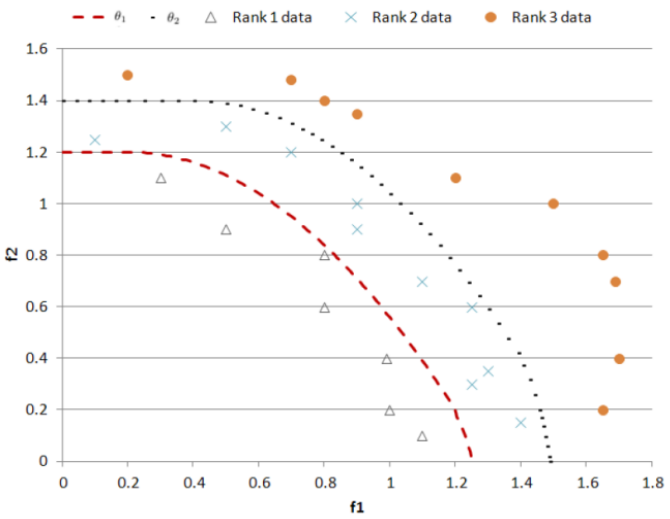
NSGA pour (Non-dominated Sorting Genetic Algorithm II) est un algorithme génétique évolutif qui permet de résoudre des problèmes multi-objectifs. Son fonctionnement se découpe en quelques grands points :

- Initialisation : L'algorithme initialise une population d'individus sur l'espace de recherche. L'objectif est de répartir uniformément afin de trouver par la suite le maximum de solutions pertinentes.

- Évaluation : L'algorithme évalue chaque individu en fonction de sa dominance et de sa "crowding distance". Pour chaque fonction objectif, les individus seront analysés pour voir quels individus représentent des solutions non dominées.

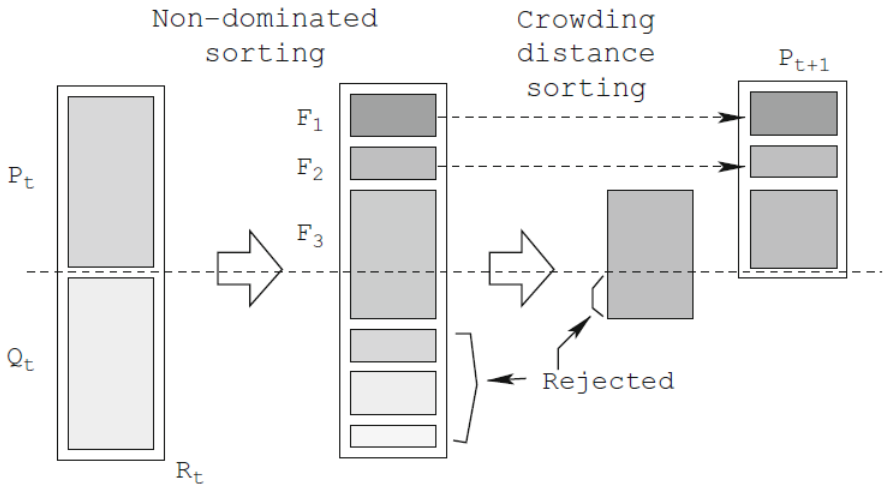
- Sélection : L'algorithme compare les individus entre eux. Les meilleurs étant sélectionnés à chaque tour de sélection. L'objectif est d'obtenir la solution la plus performante pour chaque région de notre espace de recherche. Pour ce faire, l'algorithme se base sur le rang pour juger la performance. Plus le rang est faible, plus la solution s'approche du front pareto. Afin de s'assurer que les individus soient suffisamment éloignés les uns des autres, à rang égale, la solution privilégiée est celle avec la plus grande crowding distance.

- Croisement : Les individus sélectionnés deviennent la population de la prochaine itération. Chaque individu sélectionné se reproduit et génère un second individu. Le croisement consiste à mixer les attributs de deux individus de la population parente pour générer les attributs de l'enfant.



- Mutation : Quelques enfants seront sélectionnés aléatoirement et feront l'objet d'une mutation aléatoire. Un attribut sera modifié aléatoirement pour continuer à explorer l'espace de recherche.

- Condition d'arrêt : L'ensemble de ces étapes sont répétés jusqu'au critère d'arrêt.



Les données

L'enjeu est de générer des données suffisamment compétitives et cohérentes afin de s'approcher d'une situation réelle pour pousser l'algorithme à nous donner plusieurs solutions.

Lors de nos premiers essais, nous avons généré des valeurs aléatoirement pour nos variables de décisions. Le nombre d'instructions par tâches varient de 2 000 000 et 30 000 000. La capacité des serveurs en instructions par secondes varient de 40 000 à 150 000 et le coût est proportionnel à notre capacité. L'algorithme NSGA 2 convergeait systématiquement vers une solution parce qu'un serveur était trop compétitif par rapport à ses voisins.

Nous avons conservé nos fonctions aléatoires pour générer le nombre d'instructions par tâche ainsi que la capacité des serveurs. Le coût en revanche s'exprime de la façon suivante :

Coût = Capacité * 0.00001 * X

Sachant que X suit une loi normale N(5,0.6)

Pour ce projet nous avons utilisé 20 tâches pour 5 serveurs en données fixes:

Instructions des tâches (20)	3324661,6725193,13723233,5779370,14563959,10202920,18713183,16684225,15556916,9946119,15537110,22322498,14681901,22989230,23444017,13560735,8833867,6920224,13022243,12300466
Capacité des serveurs (5)	95546,100505,44292,117400,110803
Coût unitaire des serveurs (5)	3.46478974, 5.69474934, 1.9086627 , 6.94341464,5.03096405

Développement et résultats

Nous avons travaillé sur deux environnements, nous avons travaillé sur un fichier notebook et en parallèle nous avons eu l'idée de mettre en œuvre une application interactive (site internet) à l'aide de Streamlit et Python pour résoudre notre problème de Task Scheduling.

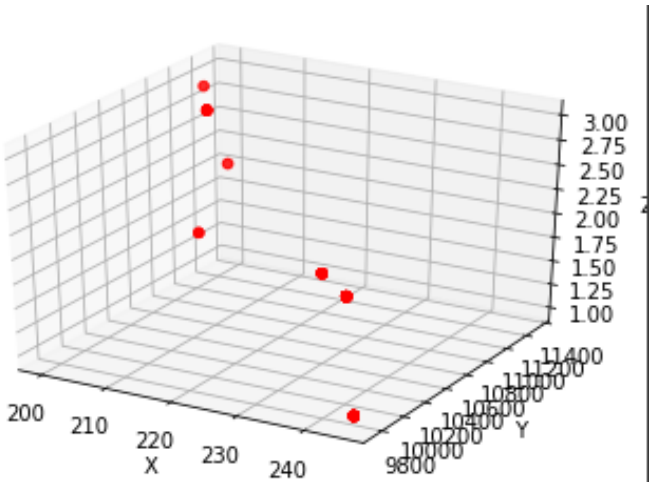
Nous avons implémenté notre algorithme NSGA II grâce à Pymoo qui est un framework open source pour l'optimisation multi-objectifs en Python

.

Sur notre jeu de données fixes évoqué précédemment à savoir :

Instructions des tâches (20)	3324661,6725193,13723233,5779370,14563959,10202920,18713183,16684225,15556916,9946119,15537110,22322498,14681901,22989230,23444017,13560735,8833867,6920224,13022243,12300466
Capacité des serveurs (5)	95546,100505,44292,117400,110803
Coût unitaire des serveurs (5)	3.46478974, 5.69474934, 1.9086627 , 6.94341464,5.03096405

Nous avons obtenu comme résultat une courbe pareto avec 7 solutions.
Voici notre courbe de pareto pour les données fixes :



Plus précisément, nous avons mis à disposition dans l'annexe,la répartition que nous propose l'algorithme pour répartir les tâches sur les différents serveurs.Ils sont également disponibles sur notre google colab.

On remarque que le 1er serveur est très compétitif. Les serveurs 4 et 5 montrent quant à eux des caractéristiques intéressantes pour traiter des tâches avec un grand nombre d'instructions.

Site web interactif

Avant de pouvoir lancer le site web interactif, on doit installer en amont les librairies requises:
pip install streamlit
pip install pymoo
pip install streamlit-plotly
pip install plotly

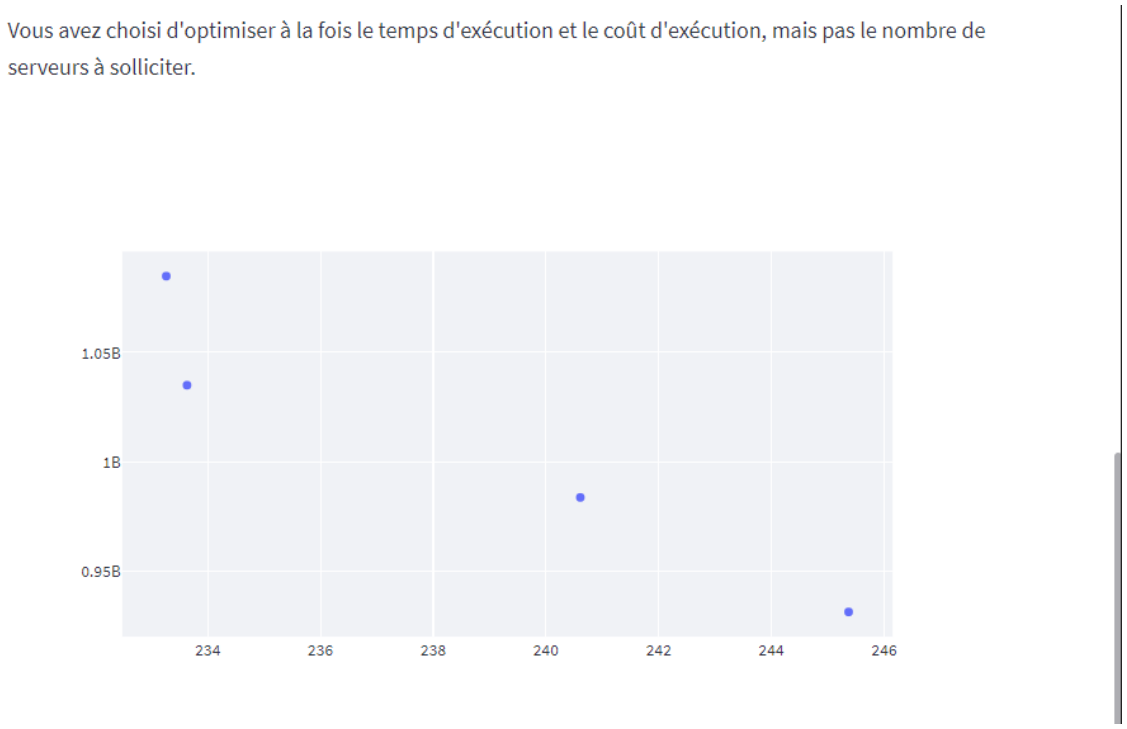
Après avoir installé toutes les dépendances, pour lancer notre page interactive, il suffit d'écrire la commande **streamlit run main.py** dans le terminal à l'endroit

ou se trouve notre fichier main.py. Puis il faut se rendre à l'adresse http indiqué. Attention streamlit n'est pas compatible avec Jupyter notebook.

L'utilité de cette application interactive est de présenter des résultats clairs et épurés. Sur cette page, nous avons rappelé les différents objectifs que l'on souhaite maximiser/minimiser et ce en utilisant le style LaTex.

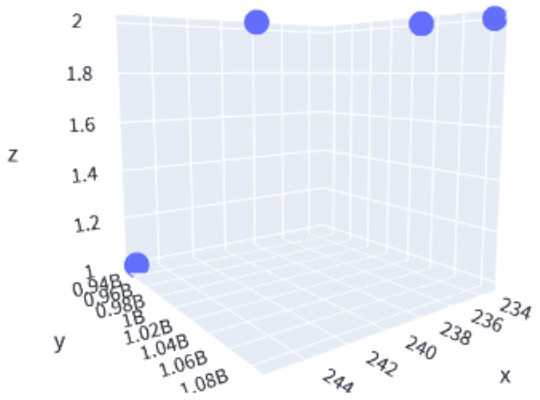
Il est possible de faire varier certains paramètres de l'algorithme comme la population, la manière de réaliser le sampling, le crossover ou encore la mutation. Il est également possible de réduire le nombre de tâches, le nombre de serveurs, choisir des valeurs "au hasard".

Il est également possible de choisir la version d'optimisation bi-objectif (minimiser le temps d'exécution et le coût d'exécution) ou la version à trois objectifs.



Vous avez choisi de maintenir les 3 objectifs.

Représentation graphique des solutions pareto



Limites et remarques

Il existe plusieurs axes d'améliorations pour notre projet.En premier lieu, ajouter l'option d'ajout de contraintes.

On pourrait également chercher à quantifier l'impact des différentes manières de réaliser le sampling, crossover, mutation. Chose qu'on a essayé de faire, mais par rapport à nos données, il n'y a pas d'impact. On pourrait également chercher à compter à partir de quel moment les solutions demeurent inchangées.

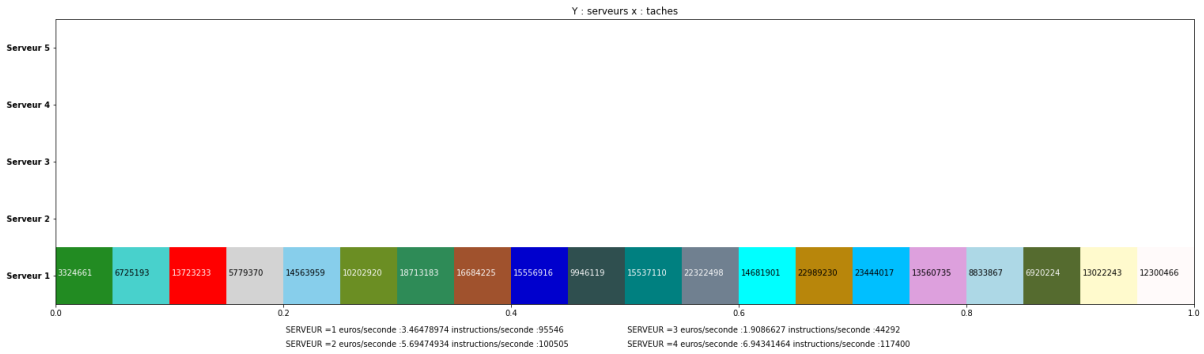
On pourrait également chercher à résoudre notre problème avec des algorithmes autres que NSGA2.

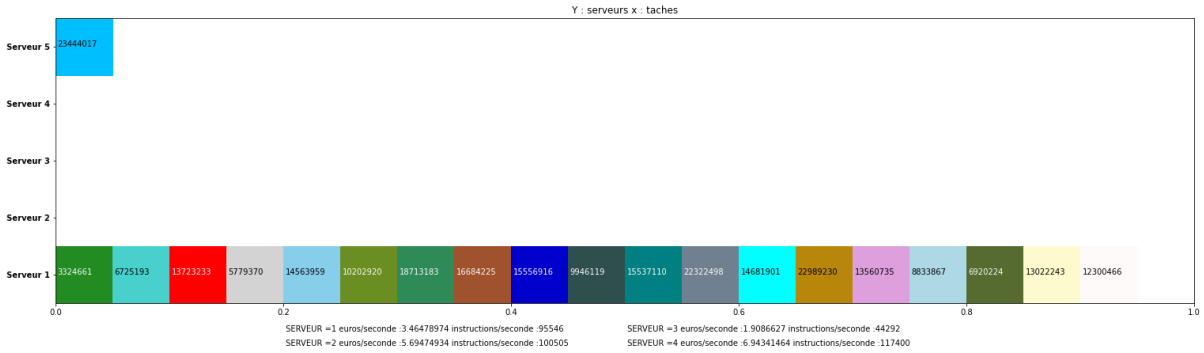
Il existe aussi des pistes innovantes qui permettent de résoudre des problèmes d'optimisation en utilisant l'intelligence artificielle comme les graphes de réseaux de neurones.

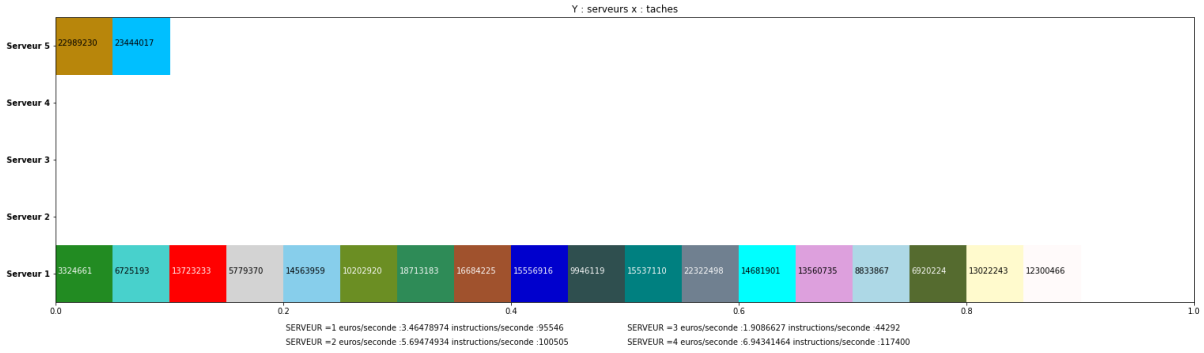
Conclusion

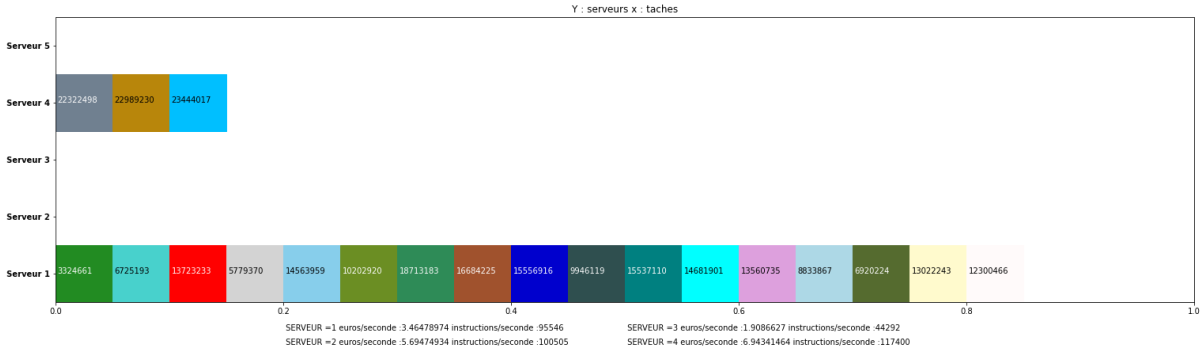
Pour conclure, nous avons eu à modéliser et résoudre un problème multi-objectif dans le domaine de Task Scheduling. Pour le faire, nous avons choisi l'algorithme NSGA II qui est l'un des algorithmes génétiques les plus performants. A l'aide de python et du framework Pymoo nous avons pu modéliser cet algorithme. Nous avons vu qu'à travers son mécanisme d'évaluation par dominance et crowding distance, sa sélection et son procédé de mutation et croisement, nous arrivons à trouver un ensemble de solutions pareto. Nous avons rempli le cahier des charges. En respectant les 3 objectifs, nous avons les meilleures solutions de répartition de tâches pour chaque serveur en fonction de son coût, de ses capacités tout en minimisant le nombre de ressources.

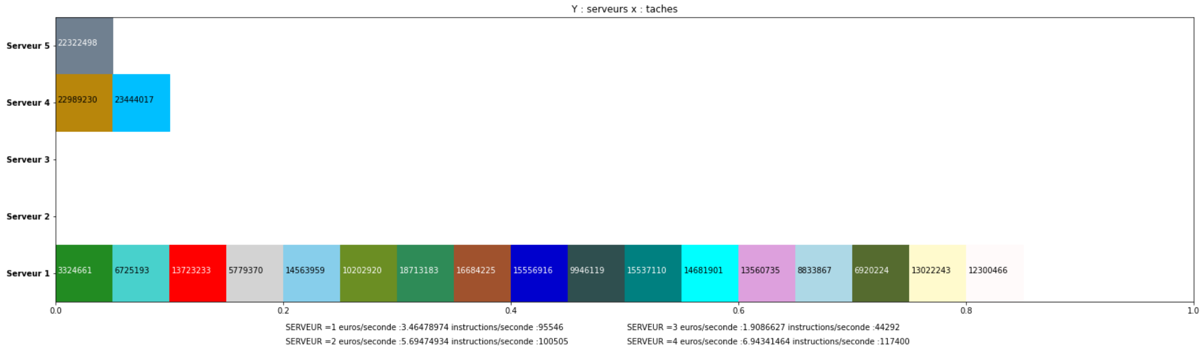
Annexes

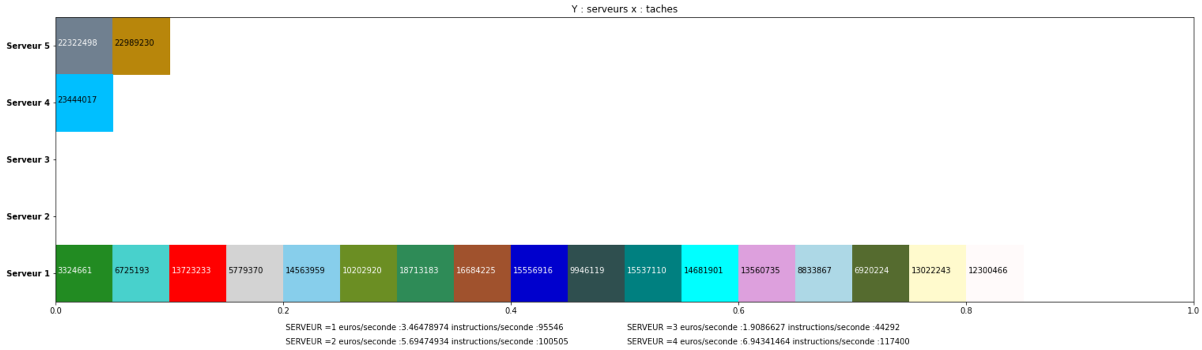

Y : serveurs x : taches
Serveur 5
Serveur 4
Serveur 3
Serveur 2
Serveur 1
3324661 6725193 13723233 5779370 14563959 10202920 18713183 16684225 15556916 9946119 15537110 22322498 14681901 22989230 23444017 13560735 8833867 6920224 13022243 12300466
0.0 0.2 0.4 0.6 0.8 1.0
SERVEUR =1 euros/seconde :3.46478974 instructions/seconde :95546
SERVEUR =2 euros/seconde :5.69474934 instructions/seconde :100505
SERVEUR =3 euros/seconde :1.9086627 instructions/seconde :44292
SERVEUR =4 euros/seconde :6.94341464 instructions/seconde :117400

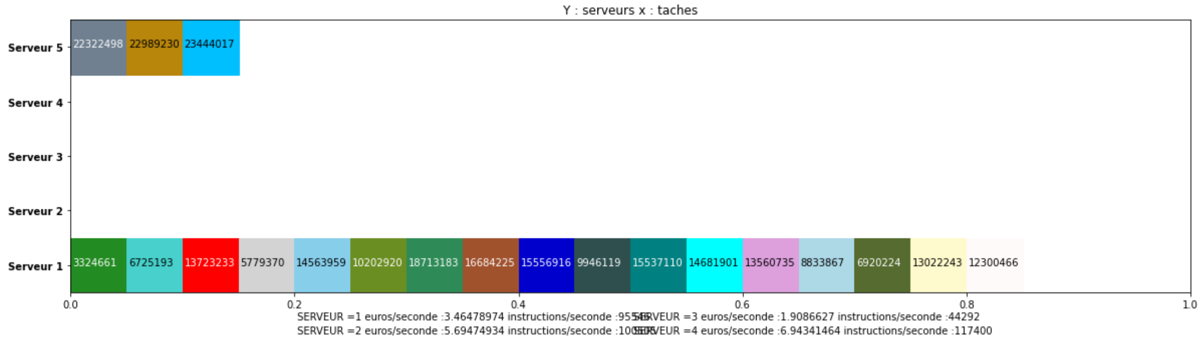

Y : serveurs x : taches
Serveur 5 23444017
Serveur 4
Serveur 3
Serveur 2
Serveur 1 3324661 6725193 13723233 5779370 14563959 10202920 18713183 16684225 15556916 9946119 15537110 22322498 14681901 22989230 13560735 8833867 6920224 13022243 12300466
0.0 0.2 0.4 0.6 0.8 1.0
SERVEUR =1 euros/seconde :3.46478974 instructions/seconde :95546
SERVEUR =2 euros/seconde :5.69474934 instructions/seconde :100505
SERVEUR =3 euros/seconde :1.9086627 instructions/seconde :44292
SERVEUR =4 euros/seconde :6.94341464 instructions/seconde :117400


Y : serveurs x : taches
Serveur 5 22989230 23444017
Serveur 4
Serveur 3
Serveur 2
Serveur 1 3324661 6725193 13723233 5779370 14563959 10202920 18713183 16684225 15556916 9946119 15537110 22322498 14681901 13560735 8833867 6920224 13022243 12300466
0.0 0.2 0.4 0.6 0.8 1.0
SERVEUR =1 euros/seconde :3.46478974 instructions/seconde :95546
SERVEUR =2 euros/seconde :5.69474934 instructions/seconde :100505
SERVEUR =3 euros/seconde :1.9086627 instructions/seconde :44292
SERVEUR =4 euros/seconde :6.94341464 instructions/seconde :117400


Y : serveurs x : taches
Serveur 5
Serveur 4 22322498 22989230 23444017
Serveur 3
Serveur 2
Serveur 1 3324661 6725193 13723233 5779370 14563959 10202920 18713183 16684225 15556916 9946119 15537110 14681901 13560735 8833867 6920224 13022243 12300466
0.0 0.2 0.4 0.6 0.8 1.0
SERVEUR =1 euros/seconde :3.46478974 instructions/seconde :95546
SERVEUR =2 euros/seconde :5.69474934 instructions/seconde :100505
SERVEUR =3 euros/seconde :1.9086627 instructions/seconde :44292
SERVEUR =4 euros/seconde :6.94341464 instructions/seconde :117400


Y : serveurs x : taches
Serveur 5
Serveur 4
Serveur 3
Serveur 2
Serveur 1
22322498
22989230
23444017
3324661
6725193
13723233
5779370
14563959
10202920
18713183
16684225
15556916
9946119
15537110
14681901
13560735
8833867
6920224
13022243
12300466
0.0
0.2
0.4
0.6
0.8
1.0
SERVEUR =1 euros/seconde :3.46478974 instructions/seconde :95546
SERVEUR =2 euros/seconde :5.69474934 instructions/seconde :100505
SERVEUR =3 euros/seconde :1.9086627 instructions/seconde :44292
SERVEUR =4 euros/seconde :6.94341464 instructions/seconde :117400


Y : serveurs x : taches
Serveur 5
Serveur 4
Serveur 3
Serveur 2
Serveur 1
22322498
22989230
23444017
3324661
6725193
13723233
5779370
14563959
10202920
18713183
16684225
15556916
9946119
15537110
14681901
13560735
8833867
6920224
13022243
12300466
0.0
0.2
0.4
0.6
0.8
1.0
SERVEUR =1 euros/seconde :3.46478974 instructions/seconde :95546
SERVEUR =2 euros/seconde :5.69474934 instructions/seconde :100505
SERVEUR =3 euros/seconde :1.9086627 instructions/seconde :44292
SERVEUR =4 euros/seconde :6.94341464 instructions/seconde :117400


Y : serveurs x : taches
Serveur 5
Serveur 4
Serveur 3
Serveur 2
Serveur 1
22322498
22989230
23444017
3324661
6725193
13723233
5779370
14563959
10202920
18713183
16684225
15556916
9946119
15537110
14681901
13560735
8833867
6920224
13022243
12300466
0.0
0.2
0.4
0.6
0.8
1.0
SERVEUR =1 euros/seconde :3.46478974 instructions/seconde :95546
SERVEUR =2 euros/seconde :5.69474934 instructions/seconde :100505
SERVEUR =3 euros/seconde :1.9086627 instructions/seconde :44292
SERVEUR =4 euros/seconde :6.94341464 instructions/seconde :117400

Démonstration de notre projet de meta-heuristiques: Task Scheduling

Enoncé du problème

On souhaite allouer un nombre de tâches définies sur différentes unités de calcul. L'objectif est de répartir la charge sur ces différents serveurs de façon à minimiser le temps d éxécution, le coût d exécution et l'utilisation des ressources.

Modélisation mathématique

🎯 Objectif n°1: Minimiser le temps d execution.

$$Min(Max(Ti)) \quad où\ Ti\ représente\ une\ ième\ tache.$$

🎯 Objectif n°2: Minimiser le cout total

$$Min(\sum_{i=1}^{n} T_i \cdot C_{vm_i}) \quad où\ Ti\ représente\ une\ ième\ tache\ et\ Cvmi\ le\ cout\ de\ la\ ieme\ vm$$

🎯 Objectif n°3: Minimiser le nombre de ressources utilisés (le nombre de serveurs)

$$Min(Rs) \quad où\ Rs\ représente\ les\ ressources.$$

Paramètres du probleme

Saisir le nombre de tâches

20

Saisir le nombre de serveurs

5

A: Valeurs par défaut | B: Valeurs au hasard

- ○ A
- ◉ B

Paramètres de NSGA2

Saisir la population souhaité

100

A: Minimiser temps déxecution et le cout déxecution | B: Minimiser temps déxecution, le cout déxecution et le nombre de serveurs à solliciter

- ◉ A
- ○ B

Sampling

- ◉ <class 'pymoo.operators.sampling.rnd.BinaryRandomSampling'>
- ○ <class 'pymoo.operators.sampling.lhs.LatinHypercubeSampling'>

Crossover

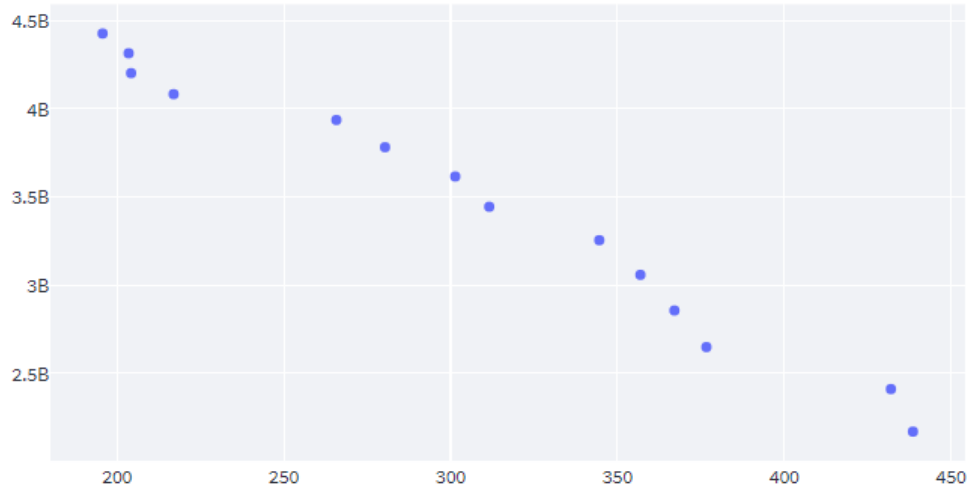
- (•) <class 'pymoo.operators.crossover.pntx.TwoPointCrossover'>
- ( ) <class 'pymoo.operators.crossover.ux.UniformCrossover'>

Mutation

- (•) <class 'pymoo.operators.mutation.bitflip.BitflipMutation'>
- ( ) <class 'pymoo.operators.mutation.gauss.GaussianMutation'>

Lancer NSGA2

Vous avez choisi d'optimiser à la fois le temps d'exécution et le coût d'exécution, mais pas le nombre de serveurs à solliciter.



Vous avez choisi de maintenir les 3 objectifs.

Représentation graphique des solutions pareto