Optimisation des paramètres d'un algorithme génétique via le problème inverse dans le cadre d'un Vehicle Routing Problem.

Stage de Master 2

Master 2 CSMI

UFR de Mathématique et d'Informatique
Université de Strasbourg

Anki Lucas

supervisé par

Alexandre Kornmann

Matéo Valdes

2020 – 2021

Table des matières

1 Remerciements

Je tiens avant tout à remercier Alexandre Kornmann ainsi que Matéo Valdes pour m'avoir proposé ce stage, pour m'avoir encadré durant celui-ci, pour avoir pu répondre à mes questions, pour avoir pu m'aider à comprendre le sujet et pour m'avoir fourni toutes les bases nécessaires permettant le bon déroulement de celui-ci. Je tiens également à remercier Michael Scherer ainsi que Neilvyn Madhia pour l'aide et les conseils qu'ils m'ont apportés lorsque j'avais des questions.

2 Introduction

Dans notre société, la prise en charge de personnes malades, blessées, ou parturientes, est devenue un enjeu important. Afin de permettre cette prise en charges, diverses sociétés spécialisées dans ce domaine ont vu le jour. Leur rôle est de fournir un transport, qu'il soit terrestre, aérien ou maritime, à ces personnes qui nécessitent une prise en charge médicale rapide, ou organisée. Cependant, l'organisation humaine et matérielle dans ce secteur est complexe, demandant beaucoup de ressources. Synovo, startup Strasbourgeoise, propose à ses clients un outil permettant de faciliter cette gestion, le logiciel Saphir.

Ce logiciel permet entre autres de générer, grâce à un algorithme, un planning optimisant les déplacements de la flotte ambulancière du client.
Cet algorithme est développé par un pôle spécifique de Synovo, le pôle Algo. Celui ci est actuellement composé d'une équipe de quatre personnes, travaillant chacune plus ou moins directement sur l'algorithme. C'est dans ce cadre que s'inscrit ce stage.

L'algorithme qui, en accédant à certaines données, va générer par la suite un planning, est un algorithme génétique, un algorithme inspiré de la théorie de l'évolution. De plus, le problème de la génération d'un planning à partir de certaines données s'appelle un Vehicle Routing Problem (VRP ou Problème de tournées de véhicules en français).

L'algorithme génétique va résoudre un VRP chaque fois qu'il est utilisé.
Cependant, l'algorithme nécessite plusieurs dizainse de paramètres, qui sont un ensemble de pondération adaptées spécifiquement à chaque utilisateur final, en fonction de ses désirs et spécificités fonctionnelles. Actuellement, ce sont les membres de l'équipe Algo qui déterminent à la main les coefficients à choisir pour chaque société de transport sanitaire. La sélection des valeurs de ces coefficients est un processus qui se déroule sur plusieurs mois, avec pour but l'obtention d'un planning correspondant aux attentes des sociétés.
De plus, dans la boucle génétique de l'algorithme, une fonction d'évaluation permet d'assigner une note à chaque planning généré. A la fin de l'optimisation, seul le meilleur planning est retenu et proposé au client. Cette fonction d'évaluation dépend également des coefficients précédents.

L'objectif du stage est de déterminer de manière automatique les pondérations, que l'on appellera coefficients de pénalisation. Cette détermination se fait en se basant sur des plannings existants que l'utilisateur à déjà fait manuellement dans le passé.

3 Synovo-Saphir

3.1 Synovo

Synovo est une startup Strasbourgeoise crée par Jérémy Wies, fondateur, directeur financier et commercial, ainsi que Michel Lacombe et Guillaume Phillip, directeurs généraux. Jérémy Wies est le directeur et fondateur de Synovo. Décidé à épauler les petites et moyennes structures dans leur développement informatique, il se lance dans un projet de création d'entreprise qui aurait vocation à simplifier la gestion du parc informatique de ses clients, leur infrastructure, leur téléphonie et l'hébergement de leurs serveurs.

C'est pour remplir ce but qu'il fonde à 18 ans la société New Web, qui comptera rapidement parmis ses clients des transporteurs sanitaires locaux, et qui s'est ensuite développée avec des offres plus généralistes.

Il fonde par la suite en 2013, avec l'aide de deux anciens camarades de classe Guillaume Philipp et Michel Lacombe, la société Synovo, concrétisant leur projet d'un logiciel de transport sanitaire. C'est en 2019 que la fusion entre Synovo et New Web, déjà très liés par avant, s'effectue, donnant naissance à Synovo Group. L'entreprise est spécialisée dans la gestion et l'optimisation de transport de santé en France depuis bientôt 8 ans. Leur cheval de bataille est la gestion logistique de flottes d'ambulances destinées aux sociétés de transport sanitaire françaises

3.2 Saphir

Saphir est le logiciel de gestion de transport sanitaire que Synovo met à disposition de ses clients. Il a été pensé pour accompagner tous les métiers d'une société d'ambulancier : régulateur, comptable, facturier, ressources humaines et personnels roulants. Ils bénéficient d'une interface dédiée afin de les aider à mener à bien leur activité avec facilité et modernité.

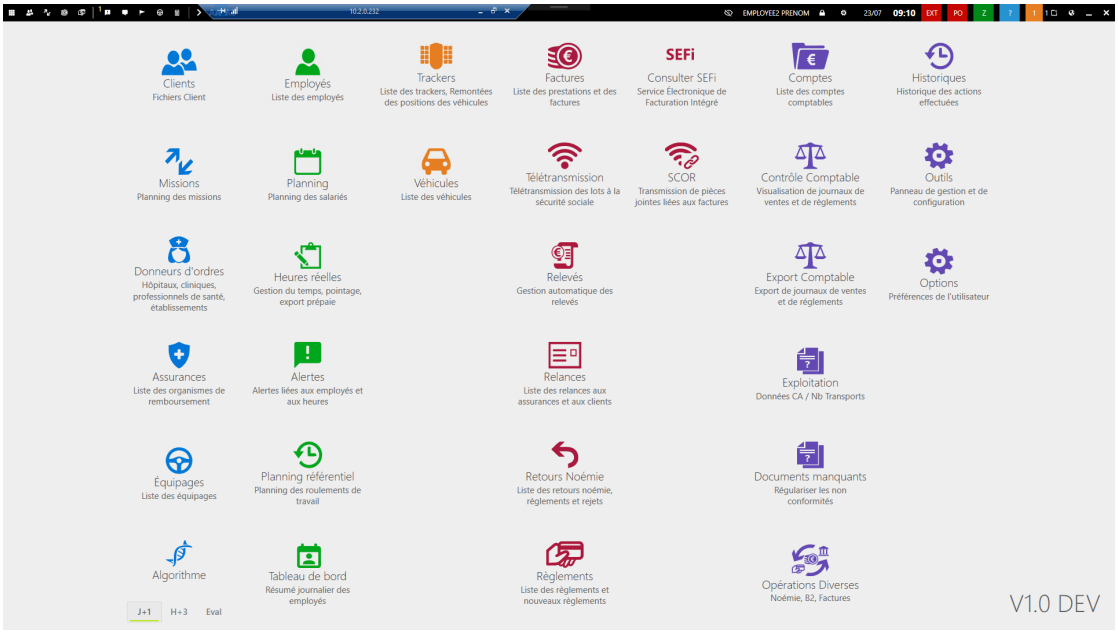

FIGURE 1 – L'interface de Saphir

Saphir ce décompose en plusieurs parties :

- La régulation des transports (Bleu)
- La gestion des ressources humaines (Vert)
- Le suivis des véhicules grâce à la géolocalisation (Orange)
- La facturation des transports (rouge)
- La comptabilité et la prépaie (Bleu)

Dans le cadre du stage, on s'intéressera uniquement aux parties bleue et verte, et plus particulièrement à la partie algorithme.
Avant de rentrer dans les détails de cette partie, expliquons un peu les fonctionnalités :

La partie sur la régulation des transports permet de tenir à jour les informations sur les clients, les missions quotidiennes, les donneurs d'ordres, les assurances et les équipages. Toutes ces informations, spécifiques à chaque société qui utilise saphir, sont maintenues à jour par des régulateurs, des secrétaires, ou des planificateurs, qui ont été préalablement formés à utiliser le logiciel.

Les ressources humaines maintiennent à jour la partie qui leur est dédiée, ce qui permet d'avoir en permanence la liste des employés ainsi que de leurs horaires et de leurs tableaux de bord.
La partie algorithme va se servir de toutes ces informations, qui sont remplies en permanence (même si l'algorithme n'est pas utilisé), afin de générer un planning pour chaque employé. Ce planning tiendra en compte, par exemple, des horaires de chaque employé, des heures de rendez-vous des patients, des véhicules disponibles selon leur type, des équipages, et d'un grand nombre d'autres paramètres qui peuvent tous être renseignés dans le logiciel (Comme par exemple la préférence d'un client pour tel chauffeur, l'autorisation d'un employé de conduire tel véhicule mais pas un autre, l'autorisation de prendre plusieurs clients en même temps ou non, etc).

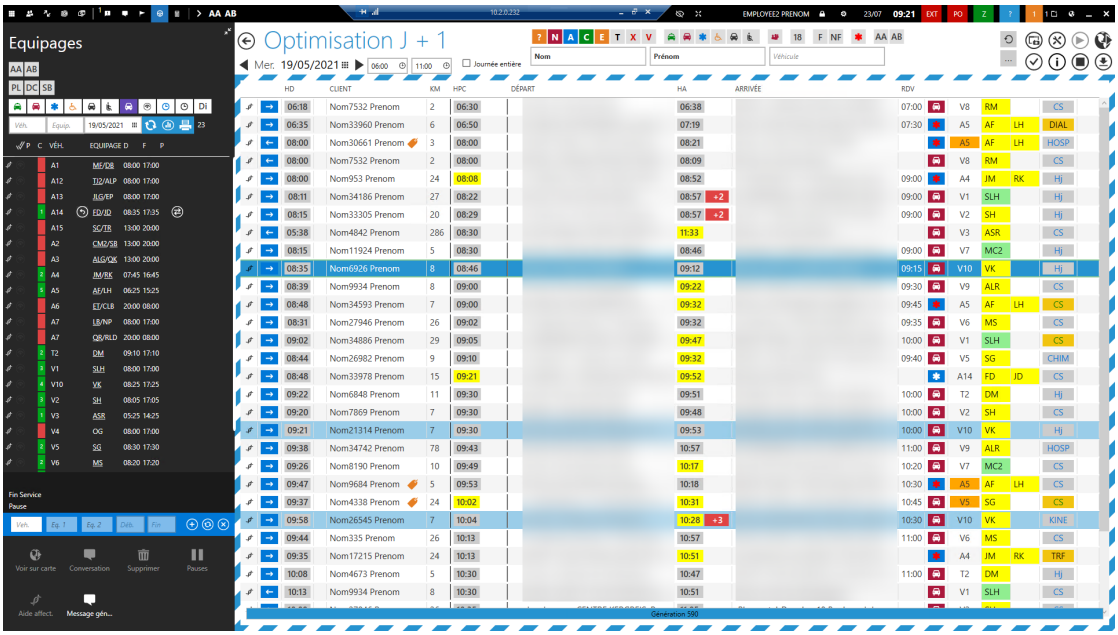

FIGURE 2 – Un planning généré par l'algorithme

Le planning tiendra donc, comme expliqué précédemment, de toutes les contraintes sur les employés, véhicules, clients, et indiquera le déroulement de la journée de chaque employé.
Nous reviendrons en détails sur le fonctionnement de l'algorithme dans une section dédiée.

4 Vehicle Routing Problem (VRP)

4.1 Le problème du voyageur de commerce

Le problème du voyageur de commerce est un problème très simple à comprendre, mais très difficile à résoudre rapidement. Son énoncé est le suivant :

Étant donné une liste de villes, et des distances entre toutes les paires de villes, quel est le plus court circuit qui visite chaque ville une seule et unique fois ?

Ce problème semble très simple pour un petit nombre de villes, mais il explose lorsque ce nombre augmente.
Par exemple d'un point de vu combinatoire, le nombre de chemins candidats pour 5 villes n'est que de 12, puis passe à 60 pour 6 villes, puis à 360 pour 7 villes. En réalité, pour n villes, le nombre de chemins possible est $\frac{(n-1)!}{2}$. Par conséquent plus le problème est grand, plus le nombre de candidats explose.

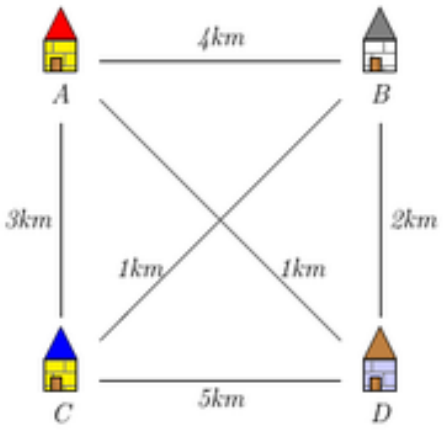

FIGURE 3 – Un exemple avec $n = 4$

Ainsi, ce problème fait partie d'une classe de problèmes appelée NP-complet, c'est à dire qu'on ne connait d'algorithme résolvant ce problème en un temps polynomial.

4.2 Vehicle Routing Problem (VRP)

Un Vehicle Routing Problem, VRP, ou encore Problème de tournées de véhicules, en francais, est une extension du problème du voyageur de commerce. Dans notre cas, il consiste à déterminer les tournées d'une flotte de véhicules, pour des interventions ou des visites médicales. Le but du problème est de minimiser le coût de ces tournées, que cela soit d'un point de vue humain, matériel, ou financier.

De notre coté, nous avons d'avantage affaire à une variante de ce problème, un Vehicles Routing Problem with Pickups and Deliveries time windows. En effet, les véhicules doivent récupérer et transporter des patients à certains endroits à une heure précise.

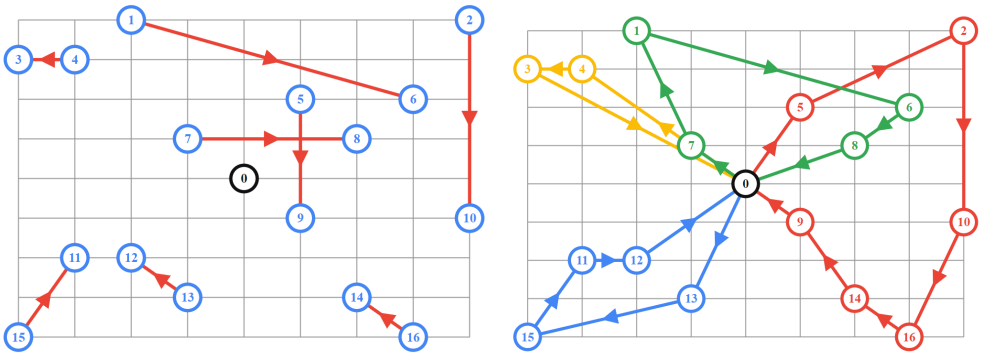

FIGURE 4 – Un Vehicles Routing Problem with Pickups and Deliveries, et une solution possible

Dans cet exemple, des véhicules doivent récupérer des objets à certains endroits et les déposer à d'autres. Il faut donc attribuer des itinéraires aux véhicules pour ramasser et livrer tous les articles, tout en minimisant la longueur de la route la plus longue. La figure de droite montre un des résultats possibles.

4.3 Un problème NP-complet

Tout comme pour le problème du voyageur de commerce, ce problème fait partie de la classe NP-complet.

Généralement, comme pour les autres problèmes de cette classe, on cherche à obtenir une solution de "bonne qualité" plutôt que la meilleure solution, afin de rester dans des temps de calculs raisonnables. Pour cela, on se tourne la plupart du temps vers des méthodes approchées heuristique.
Il existe de nombreuses méthodes heuristiques pour résoudre ces problèmes. Dans le cadre de notre VRP, c'est l'utilisation d'un algorithme génétique qui à été choisi.

5 Algorithme génétique

5.1 Algorithme génétique et boucle évolutive

Un algorithme génétique est un algorithme dit évolutionniste, c'est à dire qu'il s'inspire de la théorie de l'évolution. Les algorithmes génétiques s'inspirent de la notion de sélection naturelle en l'appliquant à une population de solutions potentielles au problème donné. Le fonctionnement d'un algorithme génétique de base est très simple. On considère des individus, solutions du problème à résoudre, que l'on génère aléatoirement. Cette population va ensuite suivre une boucle génétique dite évolutive, composée de quatre opérateurs génétiques : Évaluation, sélection, croisement et mutation.

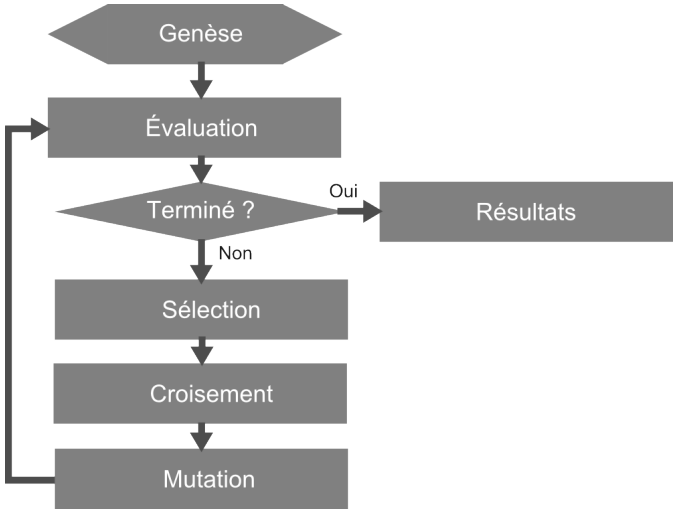

FIGURE 5 – Principe d'un algorithme génétique de base

Après création de la population initiale de manière aléatoire, la boucle d'évolution va être répétée tant que le critère désiré n'est pas atteint, comme par exemple un nombre d'itérations. Une itération s'appelle une génération.

L'évaluation est l'étape la plus importante de l'algorithme, car elle guide l'évolution de la population vers des solutions de plus en plus stables. Elle attribue un score à chaque individu, appelé fitness. Cette fonction représente donc l'espace de recherche.

La sélection consiste à choisir les individus les plus enclins à obtenir de meilleurs résultats. Cette phase est analogue au processus de sélection naturelle : Les individus les plus forts survivent, et les plus faibles meurent avant de se reproduire, ce qui favorise l'adaptation.

La phase de croisement consiste ensuite à créer de nouveaux individus, enfants, à partir de plusieurs d'individus ayant survécus appelés parents. Il existe plusieurs méthodes de croisements en fonction du type du problème, et chacune doit prendre en compte la structure des individus.

La phase de mutation permet de transformer de manière aléatoire les nouveaux individus enfants. Cela permet de favoriser l'exploration de l'espace de recherche, ou certaines solutions ne sont pas forcement accessibles avec un simple croisement. Elle permet donc, entre autre, à éviter la convergence prématurée de l'algorithme dans extremum local par exemple. Cet opérateur dépend également du type de problème.

5.2 Archived-Based Stochastic Ranking Evolutionary Algorithm (ASREA)

Un algorithme génétique classique utilise une unique fonction d'évaluation pour guider la recherche, par conséquent ils ne sont pas adaptées pour des problèmes où plusieurs objectifs doivent être optimisés en même temps. ASREA est un algorithme de type MOEA (Multi-Objective Evolutionary Algorithms)qui classe la population en comparant les individus avec les membres d'une archive contenant les meilleurs individus.

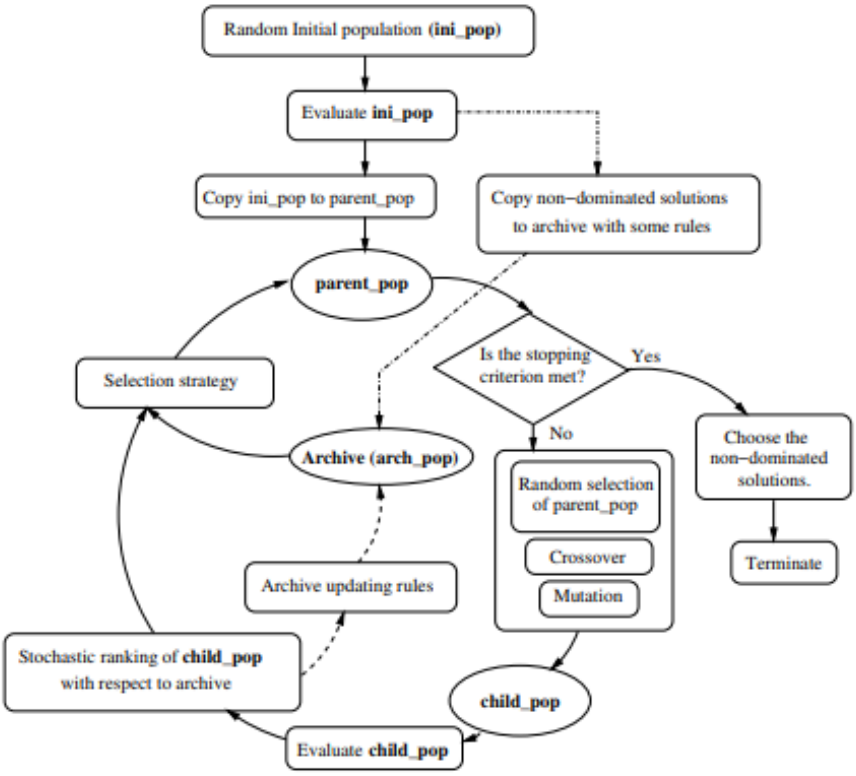

FIGURE 6 – Principe d'un ASREA

Une explication plus poussée de l'algorithme est disponible dans [1].

5.3 La fonction d'évaluation

Dans notre cas, l'algorithme génétique a pour but de fournir une solution satisfaisante pour notre VRP. Cet algorithme est guidé par sa fonction d'évaluation. Elle prend en entrée un planning, et lui assigne son fitness. Plus ce fitness est minimal, plus le planning est considéré comme bon. Ce fitness permet de sélectionner les meilleurs individus et, grâce à cette sélection, va favoriser la convergence de l'algorithme génétique vers des solutions satisfaisantes pour nous. Par conséquent au fur et à mesure des croisement et des mutations de la boucle évolutive, de nouveaux plannings (individus) seront construits à partir des anciens ayant eu un bon fitness, c'est à dire avec une valeur faible.

La fonction d'évaluation se décompose comme la somme de plusieurs fonctions de pénalisations, appelées pénalités, elles mêmes dépendant de plusieurs paramètres, appelés coefficients de pénalisation. Ces pénalités ont pour rôle de contrôler certains aspects du planning, en pénalisant certaines propriétés via des coefficients de pénalisations. Le fitness dépend donc des coefficients de pénalisation.
Par conséquent pour deux configurations différentes, c'est à dire pour deux ensembles de coefficients de pénalisations différents, un même planning n'obtiendra pas le même fitness. Donc l'algorithme génétique ne convergera pas forcément vers la même solution, et cette solution dépendra également de la valeur des coefficients de pénalisation.

6 Formulation du problème

Nous pouvons maintenant revenir sur la problématique du stage. Nous cherchons à déterminer les coefficients de pénalisation optimaux permettant à l'algorithme génétique de proposer des plannings en accord avec les critères d'un client.
Cependant un problème émerge de par la formulation de l'algorithme génétique. Comme on a pu le voir dans la partie précédente, la fonction d'évaluation se sert des coéfficients de pénalisation pour attribuer le fitness. On ne peut donc pas se servir de lui pour comparer les plannings obtenus et les plannings du client.

Dans un problème classique où la fonction d'évaluation ne dépend pas des paramètres, la méthodologie à suivre est assez simple. On considère notre algorithme génétique comme une certaine fonction F, prenant en entrée plusieurs paramètres θ_i dans $\mathbb{R}$ par exemple, et on note E la fonction d'évaluation. Le problème revient donc à minimiser cette fonction d'évaluation, c'est à dire $E(F(\theta_1, \dots, \theta_i))$.
Dans notre cas, la fonction d'évaluation dépend également des paramètres. Le problème devient donc le suivant : minimiser $E_{\theta_1,\dots,\theta_i}(F(\theta_1, \dots, \theta_i))$. Nous ne pouvons donc plus simplement regarder le résultat de la fonction d'évaluation et le minimiser, car notre fonction d'évaluation dépend, tout comme l'algorithme, des paramètres.

Pour pallier à problème, on ne va plus minimiser directement la valeur de sortie de la fonction d'évaluation, mais plusieurs valeurs de sortie de l'algorithme génétique représentant la "bonne qualité" d'un planning. Prenons par exemple le cas d'une variable intitulée Distance à vide. Cette variable, disponible en sortie de l'algorithme génétique, indique le nombre total de kilomètres pour lesquels les véhicules rouleront sans patient dans le véhicule. On peut donc se servir de cette variable intrinsèque au planning pour le caractériser.
De plus, nous disposons également de plannings pour chaque clients, représentant des exemples de plannings types qu'ils souhaitent avoir. De ces plannings types, nous pouvons également calculer les variables, tels que la distance à vide. Nous avons donc d'une part, la valeur de la distance à vide du planning type client, et d'autre part la valeur de la distance à vide provenant du planning de notre algorithme génétique. L'objectif devient donc de déterminer les paramètres permettant d'avoir une distance à vide, issue de l'algorithme génétique, la plus proche de celle issue du planning client. De la, nous pouvons reformuler le problème général de la façon suivante :

Notons Y_i^{exp} les variables représentant la bonne qualité d'un planning, issues d'un planning client. Notons Y_i ces mêmes variables, issues cette fois ci de l'algorithme génétique, de sorte que si on note F l'algorithme génétique on à :

$$\begin{pmatrix} Y_1 \\ \cdots \\ Y_k \end{pmatrix} = F(\theta_1, \dots, \theta_n)$$

On peux donc maintenant réécrire notre problème sous la forme d'un problème de minimisation des moindres carrées :

$$\min_{\theta_1,\dots,\theta_n} \sum_{i=1}^{k} \frac{(Y_i^{exp} - Y_i)^2}{\sigma}$$

Notre problème de détermination des meilleurs paramètres permettant d'obtenir le meilleur planning possible via la fonction d'évaluation s'est donc transformé en un problème des moindres carrées.

7 Méthodes d'optimisation

Pour résoudre un problème de minimisation, il est commun d'utiliser des algorithmes d'optimisation. Rappelons qu'un algorithme génétique est heuristique, par conséquent, son espace de solution est très chaotique, il possède énormément de minima locaux, ce qui rend la recherche d'un minimum global beaucoup plus compliquée et couteuse.
Par conséquent, on se tourne généralement vers des algorithmes d'optimisation globaux, plutôt que locaux, pour résoudre ce type de problème.

7.1 Simulated annealing - Dual annealing

Le recuit simulé, simulated annealing, est une méthode empirique d'optimisation, inspiré d'un processus de métallurgie consistant à alterner des cycles de refroidissement lent et de réchauffage, ce qui a pour effet de minimiser l'énergie d'un matériaux. Cette méthode a été transposée en optimisation afin de trouver les extrémas d'une fonction.

Description de l'algorithme : Par analogie avec le processus physique, on notera E la fonction à minimiser, qui sera l'énergie du système. On note T un paramètre fictif représentant la température du système. En modifiant un état donné du système, on obtient un autre état. On se retrouve donc avec deux possibilités, soit le nouvelle état améliore le critère que l'on cherche à optimiser, soit il le dégrade. Dans le premier cas, si on accepte l'état, on tend à chercher l'optimum dans le voisinage de l'état de départ. A contrario, l'acceptation d'un mauvais état permet d'explorer une plus grande partie de l'espace des états, ce qui tend à éviter de s'enfermer dans un optimum local.

A chaque itération de l'algorithme, on modifie l'état de départ, ce qui entraine une variation Δ_E de l'énergie du système. Si cette variation est négative, c'est à dire que l'énergie du système baisse, on l'applique à l'état courant. Sinon, on l'applique quand même avec une probabilité $e^{-\frac{\Delta_E}{T}}$ (On appelle cela la règle de Metropolis).
Il existe ensuite deux choix possibles pour les itérations :

- On garde la température constante, et lorsque le système atteint un équilibre, on diminue la température. On appelle ça des paliers de températures.
- On baisse la température de façon continue, comme par exemple $T_{i+1} = \lambda T_i$ avec $\lambda < 1$.

Dans les deux cas, si la température devient trop faible, ou si le système ne change plus, l'algorithme s'arrête. Notons que plus la température est haute, plus le système se déplace dans l'espace des états, ce qui a pour conséquence de choisir des états qui ne minimisent par forcément la température. Quand la température est basse, les modifications baissant l'énergie du système sont choisies, bien que d'autres peuvent l'être également, ce qui empêche l'algorithme de tomber dans un minimum local.

Voici le pseudo code de l'algorithme. On note s_0 l'état initial, $kmax$ le nombre d'itérations maximale. Si on atteint une énergie $emax$ ou moins, on arrête également l'algorithme. E est la fonction calculant l'énergie de l'état s. L'appel voisin(s) génère un état voisin aléatoire d'un état s. L'appel aleatoire() renvoie une valeur aléatoire dans $[0, 1]$. temp(r) renvoie la température à utiliser selon la fraction r du temps total déjà dépensé, et P une fonction de probabilité dépendant de la variation d'énergie et de la température.

De plus, on va considérer que s est un état optimal local, et on introduit g qui conservera l'état optimal global pour la meilleure énergie m.
Le pseudo code est le suivant :

$s \leftarrow s_0$
$g \leftarrow s_0$
$e \leftarrow E(s)$
$m \leftarrow E(g)$
$k \leftarrow 0$
while $k < kmax$ *and* $e > emax$ **do**
  $s_n \leftarrow voisin(s)$
  $e_n \leftarrow E(s_n)$
  if $e_n < e$ *or* $aleatoire() < P(e_n - e, temp(\frac{k}{kmax})$ **then**
    $s \leftarrow s_n$
    $e \leftarrow e_n$
  if $e > m$ **then**
    $g \leftarrow s$
    $m \leftarrow e$
  $k \leftarrow k + 1$
return g

Algorithme 1 : Pseudo code du recuit simulé

Le recuit double, dual annealing, consiste simplement à rajouter une étape d'optimisation locale à la fin de chaque itération du recuit simulé. Cette combinaison d'optimisation est très efficace pour les problèmes non linaires, mais en contre partie devient très couteuse en itérations et appels de la fonction à optimiser.

7.2 Optimisation bayésienne

L'optimisation bayésienne est une méthode d'optimisation séquentielle conçue pour l'optimisation de fonctions "boite noire" qui ne se représentent pas sous forme de fonctions usuelles. On l'utilise généralement pour optimiser des fonctions couteuses à évaluer.

On l'utilise généralement pour résoudre des problèmes de la forme

$$\min_{x \in A} f(x)$$

avec A un ensemble de points. De plus il y a également les contraintes suivantes :

- f est une fonction boite noire pour laquelle on ne connait pas de représentation sous forme de fonction, donc pour laquelle on ne connait pas son gradient
- f est très couteuse à évaluer
- Plusieurs évaluation de la fonction en un même point peuvent donner un résultat différent, la fonction peut être bruité

Comme la fonction à optimiser est inconnue, la stratégie bayésienne consiste à utiliser un *prior*, c'est à dire une loi de probabilité qui se fonde sur les précédentes évaluations. Cela permet d'obtenir des informations sur le comportement de la fonction.

Une fois que l'on connait plusieurs évaluations de la fonction, que l'on considèrent comme données, le prior est mis à jour pour former la loi de probabilité postérieure. Cette loi sert à construire une fonction d'acquisition qui va déterminer le prochain point à évaluer. Il existe plusieurs méthodes pour décrire les distributions antérieure/postérieure, la plus courante étant d'utiliser des processus gaussiens.

Il existe également plusieurs fonctions d'acquisitions, comme par exemple :

- Expected improvement (EI) : $-EI(x) = -\mathbb{E}[f(x) - f(x_t^+)]$
- Lower confidence bound (LCB) : $LCB(x) = \mu_{GP}(x) + \kappa\sigma_{GP}(x)$
- Probability of improvement (PI) : $-PI(x) = -P(f(x) > f(x_t^+) + \kappa)$

où x_t^+ est le meilleur point observé jusqu'à présent, κ une variable contrôlant l'exploration.

Voici le pseudo-code décrivant l'algorithme :

Placer le prior sur f
Définir la fonction d'acquisition qui, en lui donnant la distribution postérieure, va donner des nouveaux points à évaluer
Évaluer n points aléatoire de f en enregistrant les résultats
while $i <$ *Nombre maximum d'évaluation* **do**
    Déterminer la distribution postérieure à l'aide des points déjà calculés
    Trouver le nouveau point x_i à évaluer, en optimisant la fonction d'acquisition
    Évaluer $f(x_i)$ et rajouter cette évaluations aux précédentes
    Incrémenter i

Algorithme 2 : Pseudo code de l'optimisation bayésienne

7.3 Differential evolution

Un algorithme à évolution différentielle, differential evolition, fait partie de la famille des algorithmes évolutionnistes, servant à faire de l'optimisation globale en s'inspirant de la théorie de l'évolution. On appelle communément ces méthodes des métaheuristiques car elles font peu d'hypothèses sur le problème à optimiser et permettent la recherche dans un gros espace de candidats solution. Cependant, ces métaheuristiques ne garantissent pas la découverte d'une solution optimale.
A l'instar des méthodes précédentes, l'évolution différentielle ne nécessite pas que le problème d'optimisation soit différentiable. Il est très souvent utilisé dans le cadre de problème non continue et bruité.

L'évolution différentielle optimise un problème en maintenant une population de candidats solutions et en créant des nouveaux candidats en combinant les anciens, puis en conservant les candidats solutions qui ont le meilleur fitness pour le problème d'optimisation de départ. Par conséquent, on considère généralement le problème d'optimisation comme une boite noire qui nous donne simplement une mesure pour un candidat.

Description de l'algorithme : Un algorithme d'évolution différentielle fonctionne en possédant une population de candidats solutions, appelés agents. Ces agents sont déplacés dans l'espace de recherche en utilisant des formules simples permettant de combiner les positions des agents

existants de la population.
Si la nouvelle position d'un agent est une amélioration, alors on l'accepte dans la population, sinon on la rejette. Le processus est ensuite répété dans l'espoir de trouver une solution suffisamment satisfaisante.
Si l'on note $f : \mathbb{R}^n \to \mathbb{R}$ la fonction à minimiser (maximiser revient à prendre $-f$), le problème à résoudre est donc

$$\min_{x \in \mathbb{R}^n} f(x)$$

Voici le pseudo code d'un algorithme d'évolution différentielle de base :

Définir $CR \in [0, 1]$, la *crossover probability*, par exemple $CR = 0.9$
Définir $F \in [0, 2]$, differential weight, par exemple $F = 0.8$
Définir $NP \geq 4$, la taille de la population candidats, généralement $10n$
Initialiser $x \in \mathbb{R}^n$ aléatoirement
while *Condition de terminaison non atteinte (Nombre d'itérations maximale ou certaine fitness atteinte)* **do**
  for *Chaque agent x de la population* **do**
    Choisir 3 agents a,b,c aléatoirement dans la population distincts entres eux et de x
    Choisir un indice $R \in \{1, \ldots, n\}$, avec n la dimension du problème à optimiser
    Calculer la nouvelle position de l'agent de la manière suivante :
    for $i \in \{1, \ldots, n\}$ **do**
      Définir $r_i \in U(0, 1)$ avec U une loi uniforme
      if $r_i < CR$ *or* $i = R$ **then**
        Poser $y_i = a_i + F \times (b_i - c_i)$
      else
        Poser $y_i = x_i$
    if $f(y) \leq f(x)$ **then**
      Remplacer x dans la population par y
Renvoyer comme candidat solution l'agent qui a le meilleur fitness

Algorithme 3 : Pseudo code de l'évolution différentielle

7.4 Curve fitting

Le curve fitting est une technique consistant à construire une courbe à partir de fonctions mathématiques en ajustant leurs paramètres, dans le but de se rapprocher de la courbe mesurée. On utilise des méthodes de régressions pour traiter ce genre de problème, dans notre cas on considèrera la méthode des moindres carrés non linaire.

Les moindres carrés non linéaire est une forme des moindres carrés plus adaptée à l'estimation de modèle non linéaire. On cherche à déterminer n paramètres en se basant sur un nombre $m > n$ d'observations.
La méthode consiste à résoudre le problème suivant : On considère $m \in \mathbb{N}$ couples d'observations (x_i, y_i) et une fonction de régression de la forme $y = f(x, \beta)$, où β est un vecteur de $n \leq m$

paramètres. On souhaite déterminer le vecteur β résolvant le problème :

$$\min_{\beta} \frac{1}{2} \sum_{i=1}^{m} (y_i - f(x_i, \beta))^2$$

Si l'on pose $r_i(\beta) = (y_i - f(x_i, \beta))$ alors le problème devient

$$\min_{\beta} g(\beta)$$

avec $g(\beta) = \frac{1}{2} \sum_{i=1}^{m} r_i(\beta)^2$.
Pour écrire le problème quadratique pour la minimisation, il faut les dérivées premières et secondes de g(x).
La dérivée première s'écrit

$$\nabla g(\beta) = \sum_{i=1}^{m} r_i(\beta) \cdot \nabla r_i(\beta) = (\nabla r(\beta))^T \nabla r(x)$$

avec $\nabla r(\beta)$ la matrice Jacobienne. Le vecteur $\nabla r_i(\beta)$ correspond à la ligne i de la matrice Jacobienne.
La dérivée seconde s'écrit

$$\begin{aligned} \nabla^2 g(\beta) &= \sum_{i=1}^{m} (\nabla r_i(\beta) \cdot (\nabla r_i(\beta))^T + r_i(\beta) \cdot \nabla^2 r_i(\beta)) \\ &= (\nabla r(\beta))^T \nabla r(\beta) + S(\beta) \end{aligned}$$

avec $S(\beta) = \sum_{i=1}^{m} r_i(\beta) \nabla^2 r_i(\beta)$.

Il existe plusieurs algorithmes pour résoudre ce problème.

7.4.1 L'algorithme de Gauss-Newton

L'algorithme de Gauss-Newton est une modification de l'algorithme de Newton dans le cas multidimensionnel permettant de trouver le minimum d'une somme de fonctions au carré, et ne nécessitant pas le calculs de dérivées secondes. En effet, dans cette méthode, on néglige le terme $S(\beta)$ de la dérivée seconde.
L'algorithme se présente sous la forme suivante :

Choisir $\beta^{(0)}$
$\beta^{(k)} \leftarrow \beta^{(0)}$
$r^{(k)} \leftarrow y - f(x, \beta^{(0)})$
while *Critère d'arrêt non atteint* **do**
  Calculer $\nabla r^{(k)}$ le gradient de $r^{(k)}$ par rapport à $\beta^{(k)}$
  Calculer $s^{(k)}$ solution de $(\nabla r^{(k)})^T \nabla r^{(k)} s^{(k)} = -(\nabla r^{(k)})^T r^{(k)}$
  $\beta^{(k+1)} \leftarrow \beta^{(k)} + s^{(k)}$

Algorithme 4 : Méthode de Gauss-Newton

Notons que l'algorithme peut ne pas converger si le point initial est choisi trop loin de la solution.

De plus, lorsque les résidus sont grands au points de la solution, l'approximation de la dérivée seconde peut être insuffisante, ce qui a pour conséquence de ralentir ou d'empêcher la convergence.

7.4.2 L'algorithme de Levenberg-Marquardt

L'algorithme de Levenberg-Marquardt est une variante de l'algorithme de Gauss-Newton, ayant l'avantage d'être plus stable et trouvant une solution même en étant loin d'un minimum. Cependant, il peut converger moins vite pour des fonctions très régulières.

Dans cette méthode, on approxime $S(\beta)$ par une matrice diagonale μI. L'algorithme se présente sous la forme suivante :

Choisir $\beta^{(0)}$
$\beta^{(k)} \leftarrow \beta^{(0)}$
$r^{(k)} \leftarrow y - f(x, \beta^{(0)})$
while *Critère d'arrêt non atteint* **do**
  Calculer $\nabla r^{(k)}$
  Calculer μ_k Calculer $s^{(k)}$ solution de $((\nabla r^{(k)})^T \nabla r^{(k)} + \mu_k I)s^{(k)} = -(\nabla r^{(k)})^T r^{(k)}$
  $\beta^{(k+1)} \leftarrow \beta^{(k)} + s^{(k)}$

Algorithme 5 : Méthode de Levenberg-Marquardt

On remarque que dans ce cas, trouver $s^{(k)}$ revient à résoudre

$$\begin{pmatrix} \nabla r^{(k)} \\ \mu_k^{\frac{1}{2}} I \end{pmatrix} s^{(k)} = - \begin{pmatrix} r^{(k)} \\ 0 \end{pmatrix}$$

qui peut se résoudre avec une factorisation QR. De plus on choisit généralement $\mu_k = 10^{-2}$, mais on peut le choisir judicieusement, ce qui permettra à la méthode d'être très robuste.

8 Un exemple avec des fonctions simples

On va maintenant mettre en applications les différentes méthodes d'optimisations de la section précédente pour résoudre des problèmes de minimisations du χ^2 dans le cas de fonctions mathématiques plus ou moins complexes.
Cela va permettre de tester la robustesse de chaque méthode avant de les appliquer dans le cas plus complexe de l'algorithme génétique.
On va procéder de la façon suivante :

- Choisir une fonction mathématique à plusieurs paramètres, sous la forme $f(x, \beta)$
- Fixer les paramètres β
- Générer un certains nombres de couple $(x_i, y_i = f(x_i, \beta))$ (Remarque : On peut également générer des couples légèrement bruité $(x_i, f(x_i, \beta) + \alpha b$ avec α un coefficient et b suivant une loi uniforme $U(-1, 1))$
- Construire une fonction F prenant en entrée des paramètres α, et renvoyant $\chi^2 = \sum_i (\frac{y_i - f(x_i, \alpha)}{\sigma})^2$ avec sigma un paramètre fixé
- Lancer les différents algorithmes d'optimisation afin de minimiser la fonction F

On va donc appliquer la procédure précédente avec plusieurs fonctions f.

8.1 Fonction à deux paramètres

Commençons avec la fonction suivantes à deux variables

$$\begin{aligned} f : \mathbb{R} \times \mathbb{R} &\to \mathbb{R} \\ (a, b) &\mapsto \frac{1}{1 + 2^{-(t-a)b}} - \frac{1}{1 + 2^{ab}} \end{aligned}$$

avec $t \in \mathbb{R}$ un paramètre fixé.
On génère 50 points t_i repartis linéairement dans l'intervalle $[-1, 1]$ et on fixe $a = 0.45$ et $b = 100$.
On peut ensuite générer 50 observations $f(t_i, a, b)$ ainsi que 50 observations bruité $\tilde{f}(t_i, a, b)$ puis, grâce à ces observations, construire nos fonctions F et $\tilde{F}$ à minimiser :

$$\begin{aligned} F : \mathbb{R} \times \mathbb{R} &\to \mathbb{R} \\ (x, y) &\mapsto \sum_{i=1}^{50} (\frac{f(t_i, a, b) - f(t_i, x, y)}{\sigma})^2 \end{aligned}$$

$$\begin{aligned} \tilde{F} : \mathbb{R} \times \mathbb{R} &\to \mathbb{R} \\ (x, y) &\mapsto \sum_{i=1}^{50} (\frac{\tilde{f}(t_i, a, b) - f(t_i, x, y)}{\sigma})^2 \end{aligned}$$

avec $\sigma = 10^{-8}$.
On va fixer la recherches des paramètres dans l'ensemble $[0, 2] \times [10, 600]$.
Comme nous cherchons une solution dans un espace de dimension 2, il est possible d'avoir une représentation graphique de cet espace :

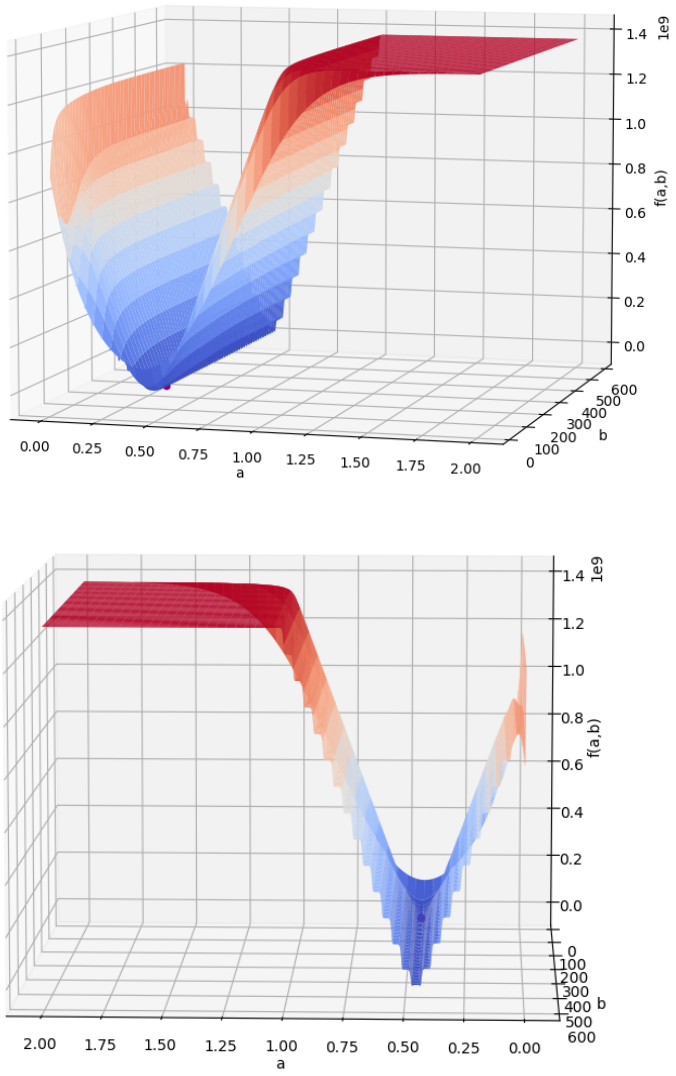

FIGURE 7 – Visualisation de la fonction F à minimiser

On constate que cet espace semble posséder plusieurs minimum locaux, ce qui le rend intéressant pour tester la robustesse des différentes méthodes.

Dans toute la suite, les fonctions d'optimisations seront issues de bibliothèques prévues pour cet effet, et on utilisera dans chaque cas les paramètres par défaut. On verra par la suite qu'il est possible d'améliorer la vitesse de convergence en changeant certains de ces paramètres.
On va donc appliquer les différentes méthodes vues dans la partie précédentes, avec leurs paramètres par défaut.

Voici les résultats obtenus avec les différentes méthodes.
Dans le cas de F :

Méthode	Converge vers	Nombre d'appel de f
Dual annealing	$(0.45, 100.0)$	4121
Optimisation bayésienne	$(0.46, 600.0)$	300
Différential evolution	$(0.45, 100.0)$	3078
Gauss-Newton	$(0.45, 100.0)$	36
Levenberg-Marquardt	$(0.45, 100.0)$	36

Dans le cas de $\tilde{F}$:

Méthode	Converge vers	Nombre d'appel de f
Dual annealing	$(0.45, 99.97)$	4088
Optimisation bayésienne	$(0.44, 158.65)$	300
Différential evolution	$(0.45, 99.97)$	552
Gauss-Newton	$(0.45, 99.97)$	36
Levenberg-Marquardt	$(0.45, 99.44)$	45

Voici ce que donnent graphiquement les approximations pour l'optimisation bayésienne :

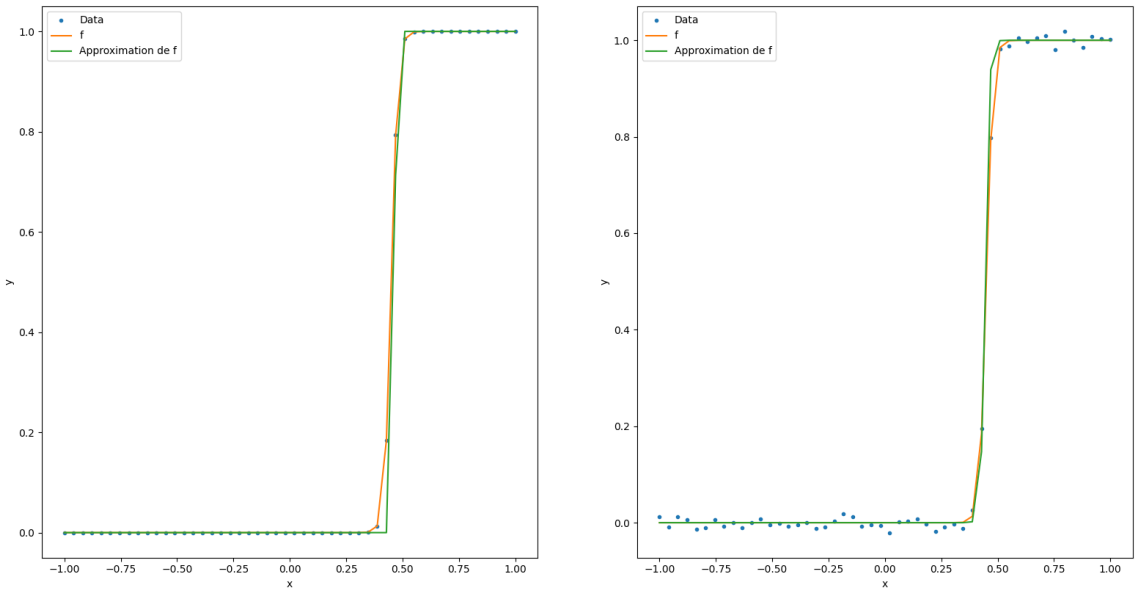

FIGURE 8 – Résultat pour l'optimisation bayesienne les données bruité (à droite) et non bruité (à gauche)

Voici la représentation pour toutes les autres méthodes qui ont des résultats similaires entre elles :

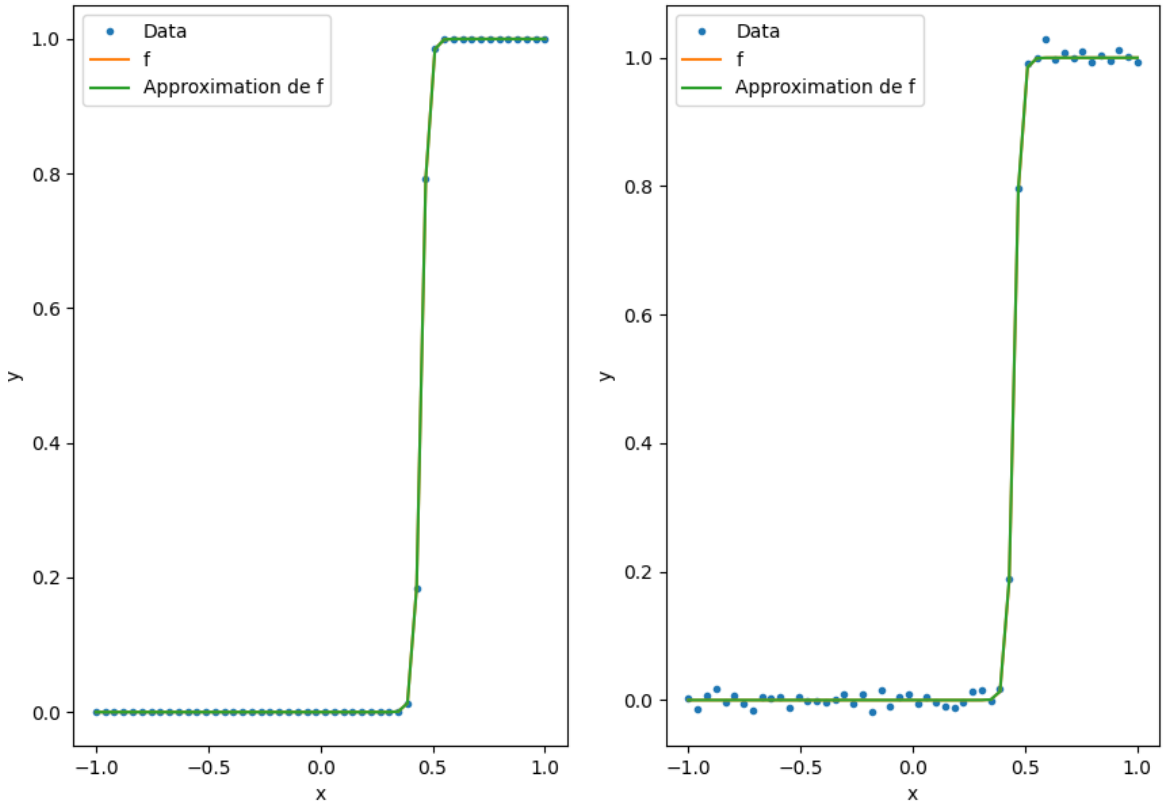

FIGURE 9 – Résultat pour les autres méthodes avec les données bruité (à droite) et non bruité (à gauche)

Analysons maintenant les résultats :

Pour toutes les méthodes, sauf l'optimisation bayésienne que l'on gardera pour la fin, on constate qu'on a bien une convergence vers les bonnes valeurs, que ça soit pour des données bruités ou non bruités. La seul chose qui diffère est le nombre d'itérations. On constate également que les méthodes les plus rapides sont les méthodes de curve fitting, avec très peu d'itérations.

Pour le dual annealing, on constate un grand nombre d'itérations, il est cependant possible de réduire nettement ce nombre d'itérations, au détriment d'un peu de précision. Pour cela on va tout simplement ne plus faire de minimisation locale à la fin de chaque étape de la minimisation globale. On part donc sur un algorithme de recuit simulé classique. De plus, on peut également limiter le nombre maximum d'évaluation de la fonction à 500 par exemple (au lieu de 10^7). Voici le résultat avec ces modifications :

	Converge vers	Nombre d'appel de f
Données non bruité	$(0.45, 100.0)$	500
Données bruité	$(0.45, 102.01)$	500

On constate qu'on obtient les mêmes résultats, avec légèrement moins de précision dans le cas des données bruitées, en limitant le nombre d'itérations et en procédant à un recuit simulé classique.

On peux également faire quelques modifications sur l'algorithme d'évolution différentielle en changeant quelques paramètres de l'algorithme. Les deux pouvant réduire significativement le nombre d'itérations sont la taille de la population, popsize, et le nombre, maxiter, de génération où on évolue la population. Au final, le nombre d'appel maximum à la fonction sera de $(maxiter + 1) \times popsize * nombre\_de\_paramètres$. En prenant donc $maxiter = 15$ au lieu de 1000 et $popsize = 10$ au lieu de 15, on arrive encore à réduire le nombre d'appel sans perdre énormément de précision :

	Converge vers	Nombre d'appel de f
Données non bruité	$(0.45, 100.00)$	356
Données bruité	$(0.45, 100.51)$	374

De plus, on passe de 3000 appels de la fonction pour les données non bruité, à 356, sans perdre de précision.

Passons maintenant au cas des résultats de l'optimisation bayésienne. Les test ont été effectués en évaluant dans un premier temps 50 points aléatoire, puis en effectuant 250 itérations de l'algorithme. La fonction d'acquisition est choisie à chaque itération de manière probabiliste entre les 3 méthodes de la partie théorique.
On constate que dans le cas bruité et non bruité, la fonction trouve le premier paramètre, mais est assez éloigné du deuxième. La raison principale peut être qu'il faut légèrement plus d'itérations pour la convergence du deuxième paramètre. De plus on constate, en regardant l'allure de l'espace à optimiser sur la figure 7, que le deuxième paramètres semble avoir beaucoup moins d'influence que le premier. En effet, tout les points de coordonnées (0.45,x) semblent être très proche du minimum. On le constate également en regardant les approximations graphiques de la figure 8 qui sont très bonnes. Enfin, si l'on a au préalable une idée de la valeur de la solution, il est également possible d'améliorer la convergence en donnant, parmi les 50 points d'initialisation, des points que l'on estime proches de cette solution.

8.2 Fonction à quatre paramètres

On va maintenant regarder le cas d'une fonction à quatre paramètres, pour voir le comportement des algorithmes dans le cas de plus grandes dimensions.
On définit la fonction

$$\begin{aligned} f : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} &\to \mathbb{R} \\ (a, b, c, d) &\mapsto ae^{-\frac{t}{b}} + ce^{-\frac{t}{d}} \end{aligned}$$

avec $t \in \mathbb{R}$ un paramètre fixé.
On procède exactement de la même façon que pour le cas à deux paramètres, en générant 50 observations t_i uniformément dans $[-20, 20]$ avec les paramètres suivants : $(a, b, c, d) = (2, 3, 4, 5)$. On va également effectuer la recherche des paramètres dans un grand espace, $[1, 100]^4$.

Comme nous avons un problème de dimension 4, il n'est pas possible d'avoir une visualisation de l'espace qui nécessiterait un graphique de dimension 5. Cependant il est toujours possible de regarder une partie de l'espace selon un plan. Voici par exemple l'allure de l'espace $(a, 3, c, 5)$ avec $(a, c) \in [1, 5]^2$:

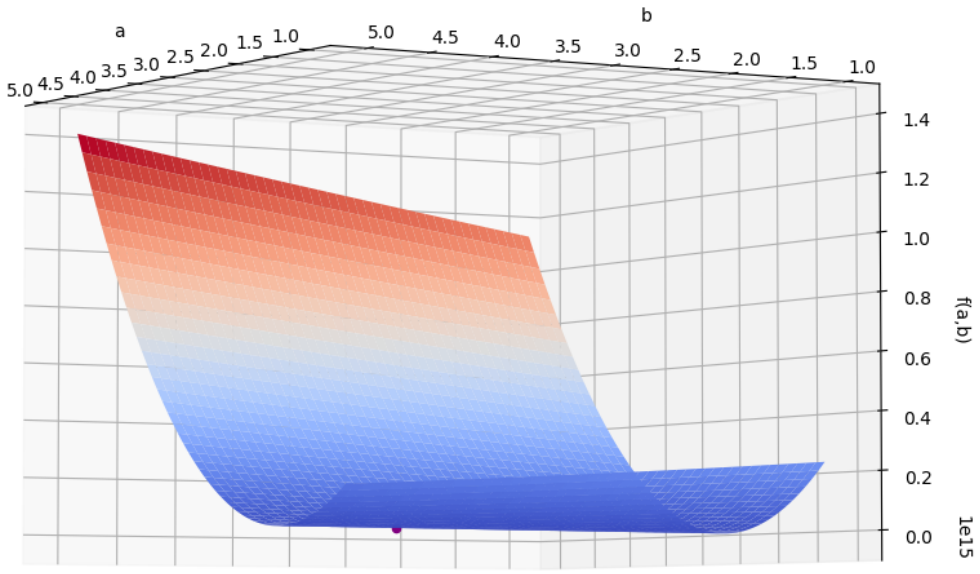

FIGURE 10 – Visualisation de l'espace $(a, 3, c, 5)$

Regardons maintenant le résultat de chacune des méthodes pour ce cas :

Sans bruit :

Méthode	Converge vers	Nombre d'appel de f
Dual annealing	$(2.00, 3.00, 4.00, 5.00)$	9346
Optimisation bayésienne	$(3.00, 88.00, 74.00, 7.00)$	300
Différential evolution	$(2.00, 3.00, 4.00, 5.00)$	39800
Gauss-Newton	$(2.00, 3.00, 4.00, 4.94)$	342
Levenberg-Marquardt	$(2.00, 3.00, 4.00, 4.94)$	337

Avec bruit :

Méthode	Converge vers	Nombre d'appel de f
Dual annealing	$(2.00, 3.00, 4.00, 4.94)$	9301
Optimisation bayésienne	$(25.00, 5.00, 81.00, 72.00)$	300
Différential evolution	$(1.95, 2.99, 3.99, 4.92)$	11270
Gauss-Newton	$(2.00, 3.00, 4.00, 4.94)$	346
Levenberg-Marquardt	$(1.95, 2.99, 3.99, 4.92)$	341

Voici ce que donne graphiquement les approximations pour l'optimisation bayésienne :

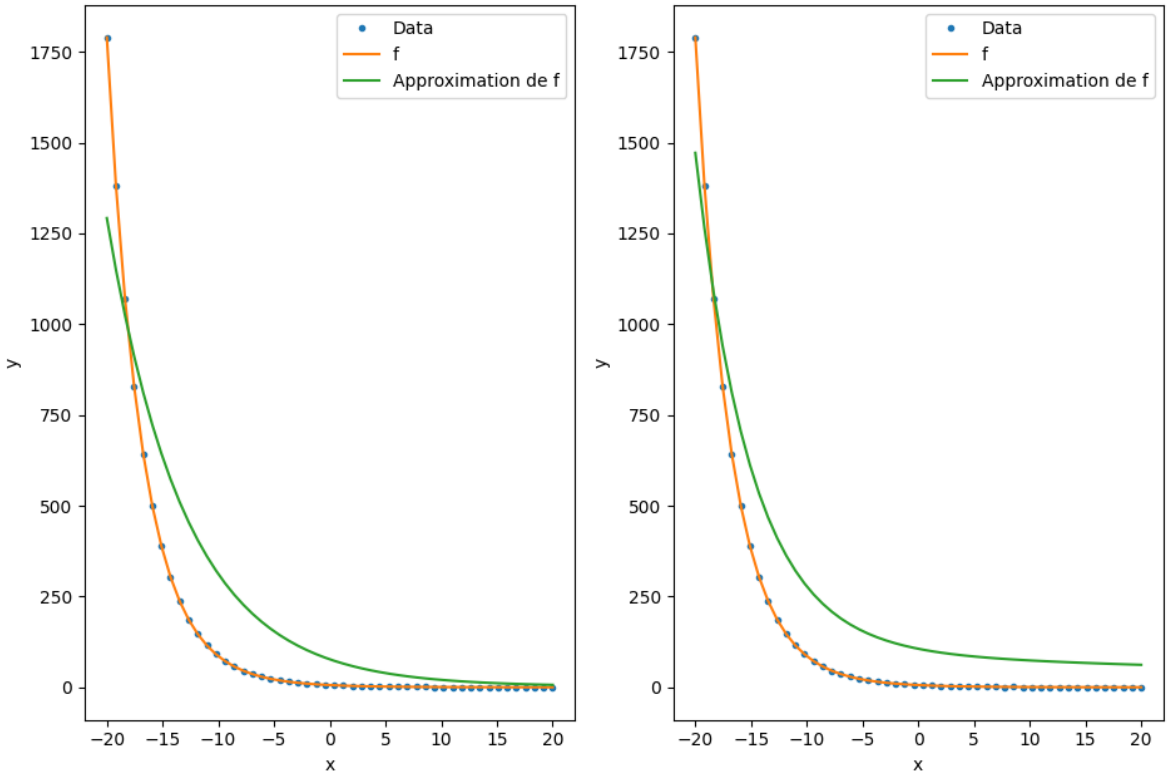

FIGURE 11 – Résultat pour l'optimisation bayesienne les données bruité (à droite) et non bruité (à gauche)

Voici la représentation pour toutes les autres méthodes qui ont des résultats similaires entres elles :

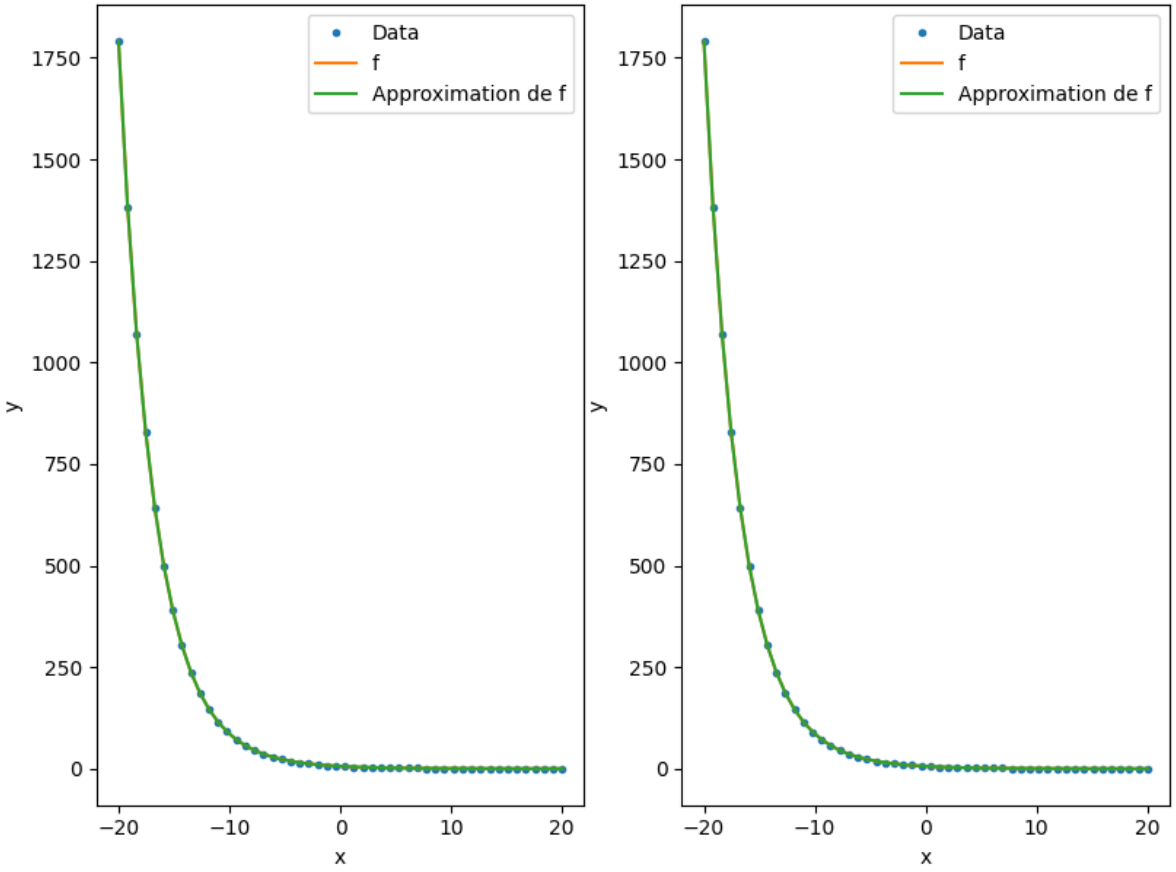

FIGURE 12 – Résultat pour les autres méthodes avec les données bruité (à droite) et non bruité (à gauche)

On constate que pour toutes les méthodes, sauf l'optimisation bayésienne dont on reviendra à la fin, on obtient un très bon résultat proche, ou même égal, à la vrai solution.
Cependant, le fait de passer de 2 à 4 paramètres a augmenter le nombres d'appel de la fonction, ce qui est normal. De plus, on a choisi de manière intentionnelle un gros espace de travail. Il est donc possible d'améliorer chaque méthode en effectuant les mêmes modifications que pour le cas à deux variables. Pour dual annealing, on va fixer le nombre d'évaluation de la fonction à 500, pour l'évolution différentielle, on va prendre $popsize = 10$ et $maxiter = 30$. Pour l'optimisation bayésienne, on va passer de 300 à 500 itérations afin d'améliorer la précision. De plus, on va réduire également l'espace de travail, on va passer de $[1, 100]^4$ à $[1, 10]^4$.

Voici les résultats en appliquant toutes ces modifications :
Sans bruit :

Méthode	Converge vers	Nombre d'appel de f
Dual annealing	$(2.00, 3.00, 4.00, 5.00)$	694
Optimisation bayésienne	$(2.71, 3.10, 3.87, 6.36)$	500
Différential evolution	$(2.00, 3.00, 4.00, 5.00)$	1710
Gauss-Newton	$(2.00, 3.00, 4.00, 4.94)$	271
Levenberg-Marquardt	$(2.00, 3.00, 4.00, 4.94)$	268

Avec bruit :

Méthode	Converge vers	Nombre d'appel de f
Dual annealing	$(2.04, 3.00, 4.00, 5.07)$	500
Optimisation bayésienne	$(2.62, 3.13, 5.31, 5.94)$	500
Différential evolution	$(1.99, 2.99, 4.02, 5.00)$	1785
Gauss-Newton	$(2.00, 3.00, 4.00, 4.94)$	291
Levenberg-Marquardt	$(1.95, 2.99, 3.99, 4.92)$	3287

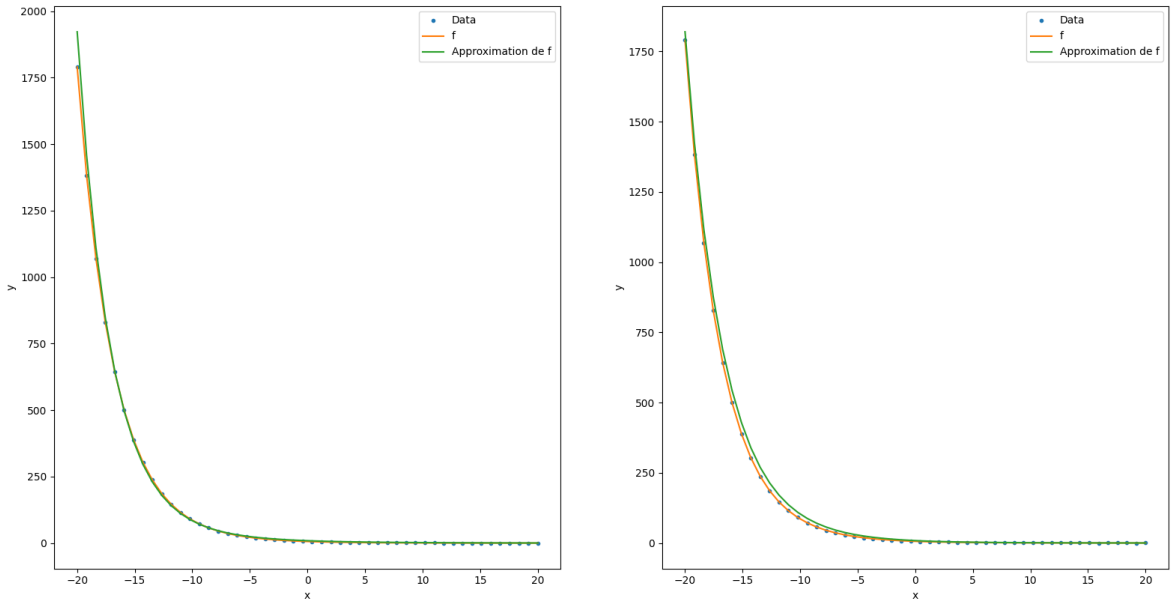

FIGURE 13 – Résultat pour l'optimisation bayesienne les données bruité (à droite) et non bruité (à gauche)

On constate pour l'optimisation bayesienne une amelioration de la précision qui se constate graphiquement. Cependant, on reste légèrement éloigné des données initiales, mais en approchant tout de même suffisamment bien la fonction.
Pour toutes les autres méthodes, on constate une baisse significative du nombre d'évaluation de la fonction, tout en gardant une très bonne précision sur la solution.

9 Application au cas de la fonction d'évaluation de l'algorithme génétique

Dans la partie précédente on a pu constater que chacune des méthodes semble converger assez rapidement, sous réserve de choisir un bon jeu de paramètres pour les algorithmes, dans le cas où la fonction permettant de construire le χ^2 est une fonction mathématique.
On va donc maintenant au cas plus général, ou cette fonction est cette fois si la sortie de la fonction d'évaluation de l'algorithme génétique.

Cependant, comme nous l'avons vu dans la partie problématique, nous faisons face à un problème majeur. On cherche à déterminer les meilleurs coefficients de pénalisation, dont dépend la fonction d'évaluation et par conséquent l'algorithme génétique, permettant d'avoir en sortie de l'algorithme génétique un planning conforme avec le planning type du client. Ce planning de sortie est celui ayant eu la note, attribuée par la fonction d'évaluation, la plus basse au cours des générations de l'algorithme génétique. Mais pour justement attribuer cette note au planning, on à besoin de coefficients de pénalisation, alors que c'est ce que l'on cherche à déterminer.

Pour contrer ce problème, on ne cherchera donc pas à minimiser directement la note, mais plutôt passer par des variables annexes intrinsèque à chaque planning et qui représentent la bonne qualité d'un planning. On pourra donc comparer ces variables pour un planning issu de l'algorithme génétique, avec ces mêmes variables issu d'un planning type client. Il nous suffira donc de minimiser l'écart entre ces variables, comme cela a été décrit dans la problématique.

Voici quelques exemples de ces variables, intrinsèque à chaque planning :

- La distance à vide : Elle représente le nombre total de mètres où des véhicules ont roulés sans patient à l'intérieur.
- La prise de service anticipée : Elle indique le nombres total d'heures où les employées ont pris leur service en avance. Par exemple, si un employé commence normalement le matin à 6h, mais qu'on lui a assigné une mission à 5h30, et qu'il doit faire 20 minutes de trajet depuis la base pour se rendre à cette mission, alors cet employé aura $20 + 30 = 50$ minutes de prises de services anticipée
- Le retard : Il représente le retard cumulé de toutes les missions

Comme nous l'avons dit, ces variables peuvent être calculées à partir d'un planning donné.
Voici donc un exemple de ces variables extraites de quatre planning type d'un client générique :

	Distance à vide	Prise service anticipée	Retard
Planning 1	168153	18900	6619
Planning 2	133677	8100	1171
Planning 3	117518	9000	3248
Planning 4	208888	12600	4954

Regardons maintenant la différence entre la plus grande et la plus petite valeur de chaque variable :

	Distance à vide	Prise service anticipée	Retard
Max-Min	91370	10800	6448

On constate que pour ces 3 variables, il y a un écart dans les ordres de grandeurs. Avec la formulation précédente du χ^2

$$\chi^2 = \sum_{i=1}^{k} (\frac{Y_i^{exp} - Y_i}{\sigma})^2$$

cet écart peut rendre certaines variables plus fortes que d'autres dans cette formulation, ce que l'on ne souhaite pas.
Par conséquent, il est important de faire en sorte de normaliser les valeurs, afin qu'aucune variable ne prenne le pas sur une autre dans le χ^2. C'est pourquoi, on cherchera désormais à minimiser un χ^2 de la forme :

$$\chi^2 = \sum_{i=1}^{k} \frac{1}{\sigma^2} (1 - \frac{1 + Y_i}{1 + Y_i^{exp}})^2$$

Revenons maintenant sur les coefficients de pénalisation sur lesquels on effectuera l'optimisation. On rappelle que ceux-ci servent de paramètres à des fonctions de pénalisation, fonctions qui servent à construire la fonction d'évaluation de l'algorithme génétique. Comme on l'a vu, il servent à pénaliser certain aspects d'un planning, ce qui permet à l'algorithme génétique de converger vers une solution acceptable pour nous.

Voici quelques exemples de ces coefficients de pénalisation :
- Distance à vide : Coefficient servant à contrôler la distance à vide .
- TapFinancial : Coefficient servant à contrôler la rentabilité d'une mission
- Retard : Coefficient servant à contrôler le retard

On constate donc que certains coefficients pénalisent des aspects que l'on a déjà évoqué dans les variables.
De la on peut donc en déduire que si l'on souhaite par exemple minimiser le coefficient de la distance à vide, il sera intéressant de construire notre χ^2 avec la variable représentant justement la distance à vide. Pour chaque coefficients de pénalisation que l'on souhaitera optimiser, on utilisera donc pour la construction du χ^2 des variables reflétant l'impact de ce paramètre sur le planning.

Maintenant que nous avons connaissance de comment exploiter nos variables en fonctions des coefficients à minimiser, nous allons procéder en deux temps.
Dans un premier temps on générera nous même un jeu données client avec des coefficients de pénalisation fixé. Le but sera de retrouver ces coefficients de pénalisation à l'aide des différentes méthodes. Ensuite, on utilisera directement des données de clients, dont on ne connait pas à l'avance la configurations de coefficients optimal, et on optimisera un certain nombre de coefficient afin d'approcher au mieux ces données.

Voici la procédure que l'on va donc suivre dans un premier temps :

- On fixe le nombre de coefficients de pénalisation à optimiser et on choisit les variables qui serviront à construire le χ^2
- On génère un certains nombre de données clients avec un jeu de coefficients de pénalisation fixé
- De ces données on y extrait les variables du chi^2 , qui seront donc nos variables d'observations

— On construit une fonction f prenant en entrée des coefficients de pénalisation, effectuant l'algorithme génétique pour chaque jeu de donnée client et pendant un certain nombre de génération, en utilisant les coefficients d'entrée, et renvoyant les variables des planning de sortie des run de l'algorithme génétique
— Construire notre fonction à optimiser, prenant en entrée les coefficients de pénalisation, appelant la fonction précédente f en récupérant les variables d'exploitation, et renvoyant le chi^2 construit à l'aide des variables d'observation et d'exploitation
— Utiliser les différents algorithmes d'optimisation sur cette fonction

Avant de commencer l'optimisation pour un paramètre, il reste un point à aborder.
Les différents coéfficients de pénalisation ont généralement des valeurs allant de 10^3 à 10^{14} pour certains, par conséquent dus à l'énorme différence d'ordre de grandeur entre ces valeurs, on ne cherchera pas à les optimiser directement. A la place, on écrira les pénalités sous la forme 10^x et on cherchera à optimiser x.

9.1 Optimisation d'un coefficient de pénalisation

On va mettre en place la procédure précédente afin d'optimiser un seul coefficient.
On génère 6 jeux de données client en fixant les coefficients de pénalisation, dont le coefficient de la distance à vide, que l'ont fixe à 10^7. On cherchera donc à retrouver cette valeur. On va utiliser 3 variables pour le chi^2 : la distance à vide (dav), le retard (ret), et la prise de service anticipée(psa).
Comme cela a été dit dans la partie précédente, on cherche à retrouver la puissance du coefficient de pénalisation sous la forme 10^x avec $x = 7$. On va donc effectuer la recherche dans l'intervalle $[3, 11]$. De plus, on arrêtera les algorithmes génétiques à 2500 itérations. Voici des extrais des résultats pour les différentes méthodes :

Coéfficient générateur	7	
Méthode	Converge vers	Nombre d'appel de f
Dual annealing	7.179	170
Optimisation bayésienne	7.007	100
Différential evolution	6.993	100
Gauss-Newton	8.191	7
Levenberg-Marquardt	9.140	5

On constate dans un premier temps que l'algorithme semble converger suffisamment proche et rapidement pour le dual annealing, l'optimisation bayésienne, et la differential evolution. Chacun de ces algorithmes a été arrêté manuellement au cours de l'optimisation, car le résultat était suffisamment proche de nos attentes dans chaque cas. En effet il faut savoir qu'une itération d'optimisation, c'est à dire six fois 2500 génération d'algorithme génétique, prenait entre 30 minutes et 1 heures.
Concernant les deux méthodes de curves fitting, aucun des test n'a permis de les faire converger. De plus, elle s'arrêtent après seulement quelques itérations. La cause est principalement le fait que pour deux run de l'algorithme génétique avec les mêmes paramètres, on peut avoir deux résultats différents. Ces méthodes ont tendance à évaluer au départ plusieurs fois des points très proches voir similaires. Par conséquent, le coté heuristique de l'algorithme fait donc qu'elles ne sont suffisamment pas adaptées pour ce problème.

Il est également intéressant de comparer les données d'observations et les données d'exploitations pour le point de convergence trouvé dans le cas des trois premières méthodes. On effectuera la comparaison pour 3 des 6 jeux de données.

Méthode	dav1	psa1	ret1	dav2	psa2	ret2	dav3	psa3	ret3
Valeurs observations	906481	16500	8926	331618	8400	1744	194852	9000	6914
Dual annealing	899535	17700	9425	337768	8400	3190	178235	9000	7900
Optimisation bayésienne	927113	17700	8680	337768	8400	3190	194852	9000	6914
Différential evolution	915529	16500	8958	331618	8400	1744	200716	9000	5922

On constate donc que pour chacune des méthodes et pour chaque jeux de données, les données d'exploitation sont très proches, voir similaires, aux données d'observations.
On va donc pouvoir passer à des cas avec plus de paramètres.

9.2 Optimisation de plusieurs coefficients de pénalisation

On parlera brièvement dans cette section de quelques tests avec 4 paramètres et plus. Avant ces tests, plusieurs tests avec deux paramètres ont été effectués. Nous n'en parlerons pas en détails ici. Cependant, ces tests ont permis d'éliminer l'algorithme differential évolution, laissant donc uniquement l'algorithme Dual annealing et l'optimisation bayésienne pour la suite.
Cette élimination est due au fait que plus le nombres de paramètres augmente, plus le nombre d'itérations requis pour obtenir un résultat convenable avec l'algorithme de differential evolution va augmenter vite en comparaisons des deux autres. Les tests à deux paramètres ont montré qu'il était donc bien en mesure de converger, mais nécessitait plus d'itérations que les autres. Sachant qu'une itérations peut très bien prendre une, voire deux heures en fonction de la quantité de données, cet algorithme n'était plus viable dans l'optique d'une optimisation avec un grand nombre de paramètre.

Maintenant que l'on a réduit l'optimisation à deux algorithmes, on va aborder quelques résultats dans le cas de plusieurs coefficients à optimiser. Chaque test prenait en moyenne une à deux semaines, en fonction de la quantité de données, ce qui limite très fortement le nombre de tests possible. De plus, suite à un incident matériel sur un des serveurs de calcul, certaines données de résultats ont été perdues. Par conséquent, on évoquera dans certains cas uniquement le résultat, sans les détails intermédiaires.

De la même façon que pour le cas à un paramètre, on va générer des données avec un certain jeux de coefficients, et on cherchera à en retrouver 4 d'entre eux. Parmi ces quatre coefficients, un est la distance à vide, un autre le delay, et les deux derniers servent à définir le "Starting Time", qui représente le cumul des heures où les employés ont commencé en avance. Ces deux derniers coefficients sont directement corrélé, ainsi par exemple un couple (x_1, x_2) peut très bien donner le même résultat qu'un couple (x_2, y_2). Ainsi on devra retrouver le quadruplé $(7, 11, 4, 2)$. On utilisera également un plus grand nombre de variables (5) pour construire le χ^2 que dans le cas précédent.

Voici quelques résultats issus de run avec l'optimisation bayesienne et Dual annealing.

Dans chaque cas, de 300 à 400 itérations des algorithmes d'optimisation ont été effectuées, et en utilisant 2500 générations pour chaque run de l'algorithme génétique :

Coéfficients générateurs	$(7, 11, 4, 2)$	
Méthode	Converge vers	Nombre d'appel de f
Dual annealing 1	$(6.90, 10.67, 5.21, 2.60)$	≈ 350
Dual annealing 2	$(8.52, 11.78, 4.68, 1.70)$	≈ 350
Dual annealing 3	$(7.39, 11.37, 3.38, 3.07)$	≈ 350
Optimisation bayésienne 1	$(3.0, 11.59, 2.0, 2.07)$	≈ 350
Optimisation bayésienne 2	$(3.0, 11.60, 3.05, 3.31)$	≈ 350
Optimisation bayésienne 3	$(6.93, 12.0, 2.0, 1.0)$	≈ 350

On constate que pour les runs 1 et 3 de dual annealing, on est très proche de la solution pour les deux premières variables, de plus, les deux dernières sont également satisfaisantes. Même constat pour le run 2, même si la première variable reste éloignée. Par conséquent, l'algorithme dual annealing semble assez bien approcher la configuration optimale de paramètre.
Pour l'optimisation bayesienne, seul le troisième des run semble être dans les horizons de la solution. On remarque que pour les autres, il semble converger vers le point $(3, 11.5, 2, 2)$. Ce point est assez éloigné de la solution, cependant il peut être intéressant de regarder l'écart entre les données observationnels et expérimentales :

Données observationnels :

$(168153, 18900, 6619, 133677, 8100, 1171, 117518, 9000, 3248, 208888, 12600, 4954, 225811, 9000, 8449)$

Données expérimentales :

$(159637, 16800, 6636, 142122, 8100, 1171, 136769, 9000, 4164, 208751, 12000, 4741, 202809, 9000, 8501)$

On constate que les données sont très proches les unes des autres. Il est donc probable que cette configuration soit un minimum locale, et approche également très bien ce jeu de données.

L'algorithme Dual annealing semble donc être celui qui ce rapproche le plus de notre solution, il sera donc conservé pour des tests avec un plus grand nombre de coefficients à optimiser. L'algorithme d'optimisation bayésienne n'est cependant pas mis de coté, car il pourrait s'avérer utile dans le cas d'une optimisation sur des données qui n'ont pas été générées à l'aide de coefficients pré établis.

10 Conclusion

L'étude préliminaire du problème à permis de retenir plusieurs méthodes d'optimisations, dont la robustesse a été testée sur plusieurs fonctions mathématiques dans le cadre d'un problème similaire. Ensuite, l'application au cas de la fonction d'évaluation pour un coefficient de pénalisation a permis de réduire le nombre de méthodes à 3.
Par la suite, les différents tests sur quatre paramètres ont permis de conserver uniquement l'algorithme dual annealing, étant celui semblant s'adapter le mieux pour notre problème. Cependant, l'optimisation bayésienne n'est pas à mettre de coté, celle-ci pouvant très bien être efficace dans le cadre de données non générées et issues directement de clients.

Des essais avec un plus grand nombre de paramètres sur des données non générées sont actuellement en train d'être effectués à l'aide de l'algorithme Dual annealing. Le but jusqu'à la fin du stage sera maintenant de mettre en application cet algorithme avec des données issues de clients dont on ne connait pas à l'avance la configurations de coefficients de pénalisation optimale.

Références

[1] Pierre Collet, *An Archived-Based Stochastic Ranking Evolutionary Algorithm (ASREA) for Multi-Objective Optimization*. Université de Strasbourg.

[2] Wikipedia, *Simulated annealing*, `https://en.wikipedia.org/wiki/Simulated_annealing`.

[3] Wikipedia, *Differential evolution*, `https://en.wikipedia.org/wiki/Differential_evolution`.

[4] Sckit-Optimize, *Bayesian optimization*, `https://scikit-optimize.github.io/stable/auto_examples/bayesian-optimization.html`.

[5] L. Vandenberghe, *Nonlinear least squares*, `http://www.seas.ucla.edu/~vandenbe/133A/lectures/nlls.pdf`.

[6] Carlos Cataniaa, Cecilia Zanni-Merka, François de Bertrand de Beuvrona, Pierre Collet, *A Multi Objective Evolutionary Algorithm for Solving a RealHealth Care Fleet Optimization Problem*. Université de Strasbourg.