Projet de synthèse

Arthur Houdant – Gaspard Dannet

IENAC21 SITA-IA

Encadrant : Logan Manen

Octobre 2023 - Fevrier 2024

Table des matières

1 Introduction

Les parkings pour avion sont des ressources importantes de l'aéroport. Ce sont des endroits où l'avion peut se garer afin de recevoir des services obligatoires au sol (comme l'embarquement ou le débarquement des passagers, l'avitaillement en carburant, l'embarquement ou le débarquement du fret, etc...). Ces ressources s'avèrent d'autant plus critiques avec l'augmentation rapide du nombre de vols (surtout dans une période post-COVID19 où le trafic aérien est très important).

Il n'existe que deux manières de palier au problème du manque de stand et de ressources face à l'expansion rapide de ces besoins.
La première consiste simplement à augmenter les installations matérielles et les ressources en équipement, telles que l'expansion de l'aéroport, l'augmentation du nombre de parkings. Cette première solution rencontre des obstacles. D'une part, l'expansion de l'aire de trafic pour y aménager des parkings supplémentaires ou encore l'investissement dans les ressources en équipement de ces parkings nécessitent un capital élevé, de la main d'œuvre, du temps et évidemment du terrain. D'autre part, cette expansion ne peut se faire éternellement et à chaque fois qu'on rencontre ce problème de pénurie de parking. Il est donc nécessaire de trouver une alternative.
La seconde manière, et celle que nous traitons dans ce projet, consiste à optimiser l'allocation des parkings pour chaque avion effectuant un séjour à l'aéroport (Gate Allocation Problem ou Gate Assignment Problem).

L'optimisation de l'allocation de ces ressources de l'aéroport permet d'améliorer l'efficacité de l'utilisation des ressources matérielles de l'aéroport et de réduire les coûts d'exploitation de l'aéroport. Actuellement, l'attribution des parkings à chaque avion dans les aéroports de grande et moyenne taille est en général basée sur une attribution manuelle fondée sur l'expérience manuelle, complétée par une attribution par le biais d'un système informatique. Dans des grands aéroports comme celui de Paris-Charles de Gaulle, il y a un flux important d'avions qui arrivent et partent de l'aéroport et ce processus se caractérise pas des délais courts et de forte densité. Un retard sur un vol pourrait entrainer, par effet "boule de neige", d'autres retards sur plusieurs autres vols. Et ces retards sont en général issus d'une mauvaise programmation des ressources aéroportuaires : selon des études statistiques, plus de 70% de tous les retards de vol causés par l'aéroport sont dus à une mauvaise programmation des ressources aéroportuaires ; 15,45% de tous les retards de vol sont des retards liés aux opérations au sol et entraînent des retards au départ des vols [1, 2].
Cela montre donc l'importance d'une bonne allocation des ressources et en particulier des parkings pour que la suite des opérations aériennes se déroule sans encombre. D'ailleurs, outre le fait d'entrainer des retards, une mauvaise programmation des ressources aéroportuaires engendrent des pertes économiques et pourraient engendrer des accidents.

En fonction de la stratégie appliquée, l'affectation d'un avion à une porte d'embarquement a un impact plus ou moins important sur les trois principales parties prenantes, à savoir l'exploitant de l'aéroport, la compagnie aérienne et les passagers. Les exploitants de compagnies aériennes sont soucieux de faciliter l'accès aux terminaux et de raccourcir les temps d'attente au sol. Les passagers recherchent la commodité en termes d'embarquement rapide et sans heurts, de courtes distances à parcourir à pied et d'accès aux équipements de l'aéroport tels que les restaurants, les aires de repos et de divertissement et les boutiques. Enfin, les opérateurs aéroportuaires veulent augmenter leurs revenus en offrant une bonne expérience aux

compagnies aériennes et aux passagers tout en maximisant l'efficacité des ressources aéroportuaires et en minimisant la congestion, les ressources nécessaires, les interruptions et les retards, etc. Contrairement à d'autres opérations de planification des compagnies aériennes, l'assignation des portes d'embarquement est par nature multi-objectifs en raison des multiples parties prenantes impliquées. Le but du GAP étant donc de satisfaire au mieux les objectifs de ces parties prenantes.

Face à ces défis complexes, la recherche de solutions efficaces pour l'optimisation de l'allocation des parkings aéroportuaires a conduit à l'exploration de diverses approches. Parmi ces approches, la programmation linéaire en nombres entiers mixtes – MIP – a été étudiée [2]. MIP offre un cadre mathématique puissant pour modéliser et résoudre des problèmes d'optimisation combinatoire, tels que le Gate Allocation Problem. Cependant, en raison de la complexité combinatoire de ces problèmes, la résolution exacte de grandes instances devient souvent difficile voire impossible en un temps raisonnable. C'est dans ce contexte que des méthodes heuristiques et métaheuristiques, comme le recuit simulé, ont émergé comme des alternatives prometteuses.

Le recuit simulé est une technique d'optimisation probabiliste inspirée du processus de recuit dans la métallurgie. Cette méthode, introduite par Kirkpatrick *et al.* dans les années 1980 [3], a montré son efficacité dans la résolution de problèmes d'optimisation combinatoire en explorant l'espace des solutions de manière probabiliste, permettant ainsi de s'échapper des optima locaux [4]. Des études ont notamment adapté avec succès le recuit simulé pour le GAP [5]. En parallèle, l'utilisation de l'intelligence artificielle a également gagné en popularité pour résoudre des problèmes complexes d'optimisation. Les techniques d'apprentissage automatique et d'optimisation basées sur l'IA ont démontré leur capacité à fournir des solutions de haute qualité dans des délais raisonnables, même pour des problèmes de grande envergure. Lam *et al.* [6] proposent une approche hybride qui combine un système expert basé sur les connaissances sous la forme d'un agent intelligent et un modèle d'optimisation tout en tenant compte des changements en temps réel des portes d'embarquement et des vols.

En mettant l'accent sur le recuit simulé dans ce projet, nous chercherons à évaluer sa capacité à fournir des solutions efficaces au problème d'allocation des parkings aéroportuaires. Le recuit simulé présente l'avantage d'être une méthode d'optimisation robuste et flexible, capable de gérer des problèmes complexes tout en évitant les pièges des optima locaux. En outre, nous évaluerons l'influence des multiples paramètres présents dans notre modèle sur la capacité à trouver une bonne solution.

Après avoir détaillé l'objectif de ce projet et la modélisation du problème, la résolution employée est expliquée. Puis les résultats obtenus en employant différents opérateurs dans le recuit simulé sont présentés.

2 Modélisation

Dans ce projet, le problème d'allocation des portes – GAP – se résume à vouloir maximiser le nombre d'avions alloués à un parking.

Cela en maximisant également le nombre d'avions possibles aux parkings dits "contacts", *i. e.* des parkings se trouvant à proximité du terminal, favorisant ainsi l'accès par les passagers et facilitant la charge de l'appareil.
Les contraintes associées à ce problème d'optimisation sont multiples. On retrouve la contrainte de comptabilité envers les parkings. Chaque parking ne permettant pas d'acceuillir tous les types avions. Il y a également une contrainte de planning. Un parking ne peut pas être utilisé par plusieurs avions simultanément. De plus, il y a une contrainte concernant le planning en ajoutant un temps supplémentaire, servant de mémoire tampon, entre la fin d'un séjour et le début d'un autre. Cette nouvelle contrainte permet de renforcer le système dans le cas de retard de précédents avions.

De plus, certains séjours ont la possibilité d'être tracté, il est ainsi possible de le découper en plusieurs opérations. Si le séjour n'est tractable qu'une seule fois, alors les deux opérations qui en découlent ont comme préférences d'être alloué à un parking contact. Cependant, si le séjour est tractable deux fois, trois nouvelles opérations sont crées. Et la deuxième opération, celle correspond au mileu de séjour, ne nécessite pas un accès direct au terminal car l'avion n'a pas à être chargé en marchandise ou passager. Il peut donc être attribué à un parking "large".

La fonction objectif utilisée dans ce projet est définie de la façon suivante : si un avion n'est pas alloué alors on augmente la finesse de 5 et si un avion est attribué à un parking "large" au lieu d'un parking "contact" alors la finesse est acrrue de 1. De cette façon le fait de ne pas être alloué à un parking est beaucoup plus impactant que d'être alloué à un parking qui est loin du terminal. Ainsi, cela favorise le nombre total d'avion attribué, mais également le taux de parking au contact utilisé.

Maintenant que la description du problème est faite et les objectifs définis, la résolution va être expliquée.

3 Résolution

L'algorithme utilisé pour résoudre ce problème est le recuit simulé qui va être détaillé dans ce chapitre. Ainsi que les différents opérateurs, définis pour ce problème spécifique, qui sont utilisés par cette méthode.

3.1 Recuit Simulé

Le recuit simulé – Simulated Annealing en anglais – est une généralisation de la méthode Monte-Carlo. Son but est de trouver une solution optimale pour un problème donné en utilisant un facteur de hasard. L'algorithme a comme paramètres initiaux une solution et une température. Le principe va être de générer à chaque itération une nouvelle solution dans le voisinage de la solution courante. Si cette nouvelle solution améliore la fonction objectif (ou énergie du système) alors elle est directement acceptée. Sinon, elle n'est acceptée qu'avec une probabilité égale à $e^{\frac{-\Delta E}{T}}$ – dans le cas d'un problème de minimisation. Avec $\Delta E = E_{courante} - E_{nouvelle}$ et $E_{courante}$, $E_{nouvelle}$ étant respectivement les énergies – *i. e.* valeurs de la fonction objectif – pour la solution courante et la nouvelle solution calculée à partir de la solution courante. Puis, à la fin de chaque itération, on décroit la valeur de la température T. La loi de décroissance généralement utlisée est $T_k = \alpha \cdot T_{k+1}$ avec $0.8 \leq \alpha \leq 0.99$. On réitére ces étapes et l'algorithme s'arrête lorsque la température atteint une certaine valeur, ou bien que le nombre maximale d'itération est atteint. Le pseudo code du recuit simulé est détaillé à l'algorithme 1.

Algorithm 1 Simulated Annealing

Require: x_{init}, T_0, α
  $x \leftarrow x_{init}$ ▷ x représente l'encodage de la solution courante
  $T \leftarrow T_0$
  $E \leftarrow$ Energy of the initial solution ▷ E est l'energie, finesse, de la solution
  while $T \geq T_{min}$ **do**
    $x_{new} \leftarrow$ Generate a solution from the neighborhood of x_{init}
    $E_{new} \leftarrow$ Energy of x_{new}
    $\Delta E \leftarrow E_{new} - E$
    if $\Delta E \leq 0$ **then**
      $x \leftarrow x_{new}$
      $E \leftarrow E_{new}$
    else
      $r \leftarrow random(0, 1)$
      if $r > e^{\frac{-\Delta E}{T}}$ **then**
        $x \leftarrow x_{new}$
        $E \leftarrow E_{new}$
      end if
    end if
    $T \leftarrow \alpha \cdot T$
  end while

A l'initialisation, la température T est généralement très grande par rapport aux différences de valeurs de la fonction objectif. Ainsi, beaucoup de solutions – mêmes celles dégradant la fonction objectif – sont acceptées. Mais, à mesure que le

nombre d'itérations augmente, la température baisse, et donc la plupart des solutions augmentant la fonction objectif sont réfusées. L'algorithme se ramène donc à une amélioration itérative classique.

Grâce au paramètre de température, le recuit simulé permet, au début de son lancement, de sortir d'un optimum local dans le but de trouver l'optimum global. Ce qu'une méthode classique d'optimisation telle que la méthode de descente de gradient ne peut pas faire. Cela permet donc d'améliorer la capacité de l'algorithme à approximer l'optimum global. Cependant, le recuit simulé peut quand même se retrouver piégé dans un optimum local à basse température. Mais il est possible de monter brusquement T pour forcer l'algorithme à continuer d'explorer. Le principal incovénient du recuit simulé réside dans le choix des paramètres. La température est intriquement liée aux valeurs de la fonction objectif. Si elle est trop basse, la qualité de la recherche sera mauvaise. Si elle est trop haute, le temps de calcul sera trop élevé. De plus, il est impossible de savoir si la solution obtenue est l'optimum global.

3.2 Représentation de la solution

Dans ce problème d'allocation de parkings, il est nécessaire de modéliser une configuration efficacement car elle sera utlisée à chaque itération. La représentation choisie est celle d'un tableau ayant comme indice les différents séjours des avions. Et l'élément aux différents indices du tableau représente le parking alloué à cet avion. Ainsi, selon ce codage, une configuration d'allocation est donnée sous la forme d'un tableau ayant comme dimension le nombre de séjours à allouer, où chaque élement du tableau contient une unique valeur représentant le parking attribué. Pour vérifier la validité de la solution selon les contraintes définies partie 2, il est nécessaire de décoder la représentation de la solution. Cette étape de décodage permet de vérifier les contraintes (contraintes de planning, contraintes de parkings compatibles), et est nécessaire à chaque itération. Cette partie de décodage est cruciale et est souvent la plus longue à réaliser du fait qu'il faut parcourir tout le tableau.

3.3 Opérateurs

Le recuit simulé est basé sur une recherche locale de la solution. Comme décrit précédemment, une nouvelle solution est obtenue à partir d'une autre solution. Il est donc nécessaire de modifier aléatoirement une solution pour obtenir une nouvelle solution dans son voisinage. Pour ce faire, on utilise des opérateurs qui vont à partir d'un solution donnée, calculer une nouvlle configuration.

3.3.1 Mutate

Le premier opérateur présenté est un opérateur classique pour le recuit simulé. Il consiste à choisir aléatoirement un indice dans le tableau représentant la solution et de changer sa valeur aléatoirment parmi la liste de ses parkings compatibles. Cet opérateur très basique ne nécessite pas beaucoup d'opérations et est donc rapide. Du fait de ne modifier qu'une valeur parmi la solution, la nouvelle configuration obtenue est très proche de l'ancienne solution. On peut envisager que cet opérateur mette beaucoup de temps à trouver une solution intéressante mais arrive facilement à partir d'une bonne solution à trouver la solution optimale.

3.3.2 Randomize Subset

Le deuxième opérateur détaillé repose sur le même principe que l'opérateur Mutate précédent. Mais la modification aléatoire ne sera pas sur une valeur, mais sur un sous-ensemble de la solution. Deux indices sont choisis aléatoirement et toutes les valeurs entre ces indices sont modifiées aléatoirement. Cette opérateur ressemble fortement à Mutate mais le voisinage d'une solution avec cet opérateur est beaucoup plus grand qu'avec Mutate. Ainsi, il est plus facile d'explorer l'ensemble de recherche des solutions. Mais, au contraire, il est plus difficle d'effectuer une recherche locale. Cet opérateur est donc utile pour fortement perturbé la solution.

3.3.3 Swap

L'opérateur Swap, consiste à échanger deux éléments dans le vecteur de la solution. Lorsque que l'un des séjours concerné n'est pas alloué, alors on choisi aléatoirement un parkings parmi ces parkings compatibles.

3.3.4 Non alloc

Cet opérateur a pour but de modifier toutes opérations qui ne sont pas allouées. Il parcourt toute la solution à la recherche, et lorsque qu'un élement n'a aucun parking attribué, l'opérateur alloue aléatoirement un parking parmi ses parkings compatibles. Toutefois cet opérateur ne modifie pas les séjours qui sont déjà alloués. Ainsi, une fois que toutes les opérations sont allouées, cet opérateur n'a plus d'utilité. Une amélioration de cet opérateur est défini dans la partie suivante.

3.3.5 Non alloc and contact

Cet opérateur est une amélioration de l'opérateur "Non alloc". En plus de modifier les opérations qui ne sont pas allouées. Cet opérateur modifie également les opérations qui sont attribués sur des parkings "large" au lieu d'un parking "contact". Ainsi cet opérateur ne va pas se retrouver bloqué comme "Non alloc", et va continuer d'essayer d'améliorer la solution. Cependant, cet opérateur ne permet pas de modifier une opération qui est allouée à un parking "contact" et donc ne permet pas d'explorer d'autres solutions. Il est donc fort probable de se retrouver dans un optimum local en utilisant ce type d'opérateur.

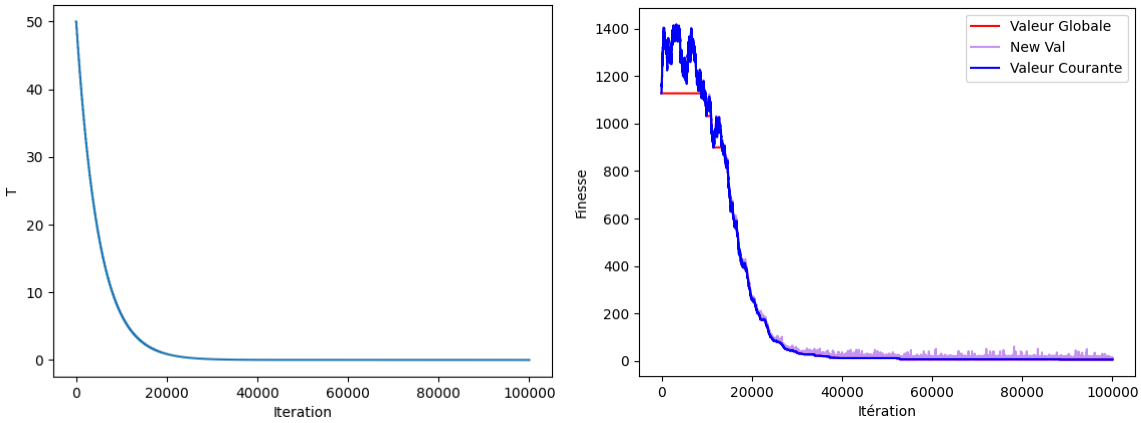
4 Résultats

Les résultats montrés dans cette partie sont réalisés soit sur la journée du 26 juin 2016 représentant 373 séjours différents – qui ont été séparés en 578 opérations. Soit sur la semaine du 20 juin au 26 juin 2016, répresentant en tout ? ? séjours et ? ? opérations. Le nombre de parkings utilisés dans les deux différents cas est de 145.

4.1 Opérateur

Dans cette partie sont affichés les résultats des différents opérateurs selon les différents données.

4.1.1 Mutate



(a) (b)

FIGURE 2 – Résultat pour la journée du 26 juin 2016 en utilisant l'opérateur Mutate

La figure 2a représente la valeur de la température en fonction du nombre d'itération. La température initiale est fixée à 50, et $\alpha = 0.98$. De plus, 100 itérations sont calculées pour une même température T. Avec cet opérateur Mutate, les résultats obtenus pour la journée du 26 juin 2016 sont montrés figure 2b. La finesse représentée en ordonnée correspond à la valeur de la solution par la fonction objetif. La solution initiale a pour finesse 1150. On observe que la valeur courante au début de l'exécution a la possibilité de monter. Cela est dû à l'impact de la température qui est assez élevée pour permettre l'algorithme d'explorer des solutions moins pertinentes. Cependant, plus la température diminue, plus la valeur courante diminue et tend à converger. Dans ce cas là, on obtient comme valeur globale à la fin de l'algorithme un score de finesse de 5, qui correspond à cinq opérations qui ont été alloués sur des parkings large. Pour cette exécution, avec l'opérateur Mutate, avec 100000 itérations, il a fallu 66 secondes pour terminer l'éxecution.

4.1.2 Randomize Subset

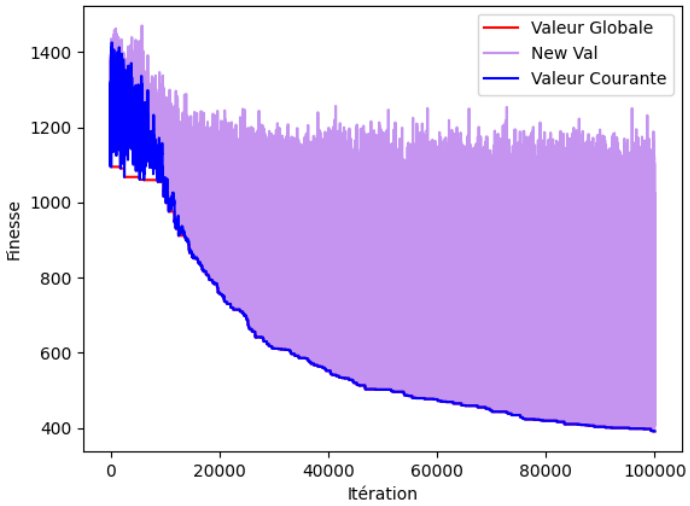


FIGURE 3 – Résultat pour la journée du 26 juin 2016 en utilisant l'opérateur Randomize Subset

La figure 3 représente l'évolution de la valeur globale et courante. La loi de décroissance de la température est la même que celle montrée figure 2a. En plus de la valeur courante et globale, la valeur calculée à chaque itération, obtenue grâce à l'opérateur sur la solution courante est affichée. On constate que cet opérateur a pour effet de rechercher une nouvelle solution dans un voisinage très large de la solution courante. Ce qui entraîne une performance mitigée. Cela peut être efficace au début de l'algorithme pour trouver une bonne solution, mais il va être difficile de converger vers la solution optimale. Comme on le voit sur la figure 3, au bout de 100000 itérations la finesse obtenue est de 391 pour une durée de 106 secondes.

Cet opérateur, contrairement au précédent, semble mettre plus de temps à converger vers une bonne solution. On peut remarquer que la courbe tend à décroitre encore même après 100000 itérations. Il peut donc être tentant de tester ce qu'il en est avec plus d'itérations.

4.1.3 Swap

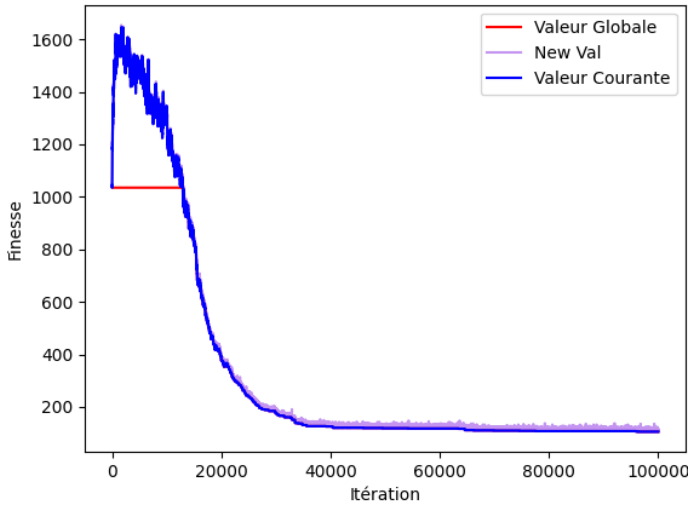
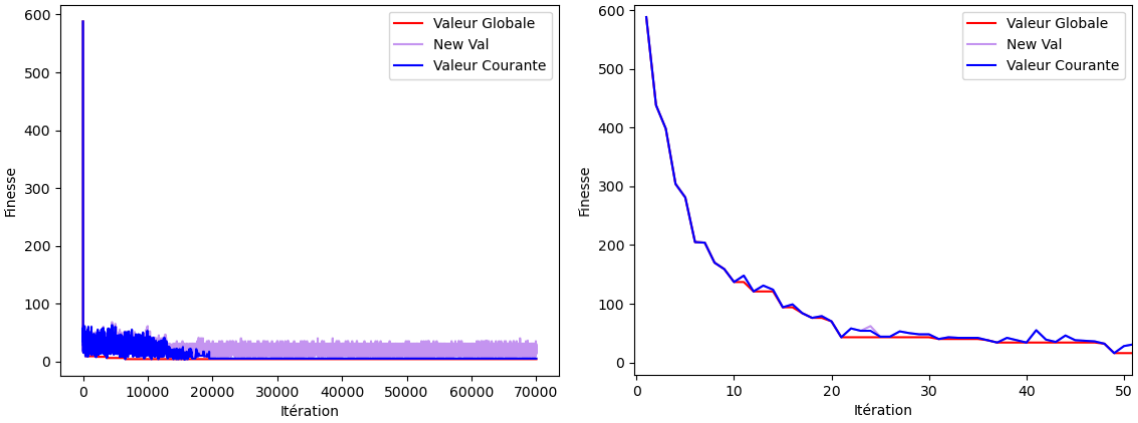


FIGURE 4 – Résultat pour la journée du 26 juin 2016 en utilisant l'opérateur Swap

La figure 4 représente l'évolution de la valeur globale, de la valeur courante ainsi que les nouvelles solutions obtenues à chaque itération. La loi de décroissance de la température est la même que celle montrée figure 2a. Cette courbe a le même profil que celle de l'opérateur Mutate sur la figure 2b. Elle accepte beaucoup de solutions au début, pour des températures basses, puis converge vers une solution. Cependant, Swap est moins performant que Mutate. En effet, pour une durée de 78 secondes, Swap permet de passer d'une valeur initiale de 1050 à 106 contre seulement 5 pour Mutate. De plus, contrairement à Randomize Subset, Swap converge déjà en 100 000 itérations et ne semble pas pouvoir grandement améliorer la solution. Il peut être intéressant de voir si, combiné à d'autres opérateurs, Swap se révèle être une bonne méthode pour atteindre une bonne solution.

4.1.4 Non alloc

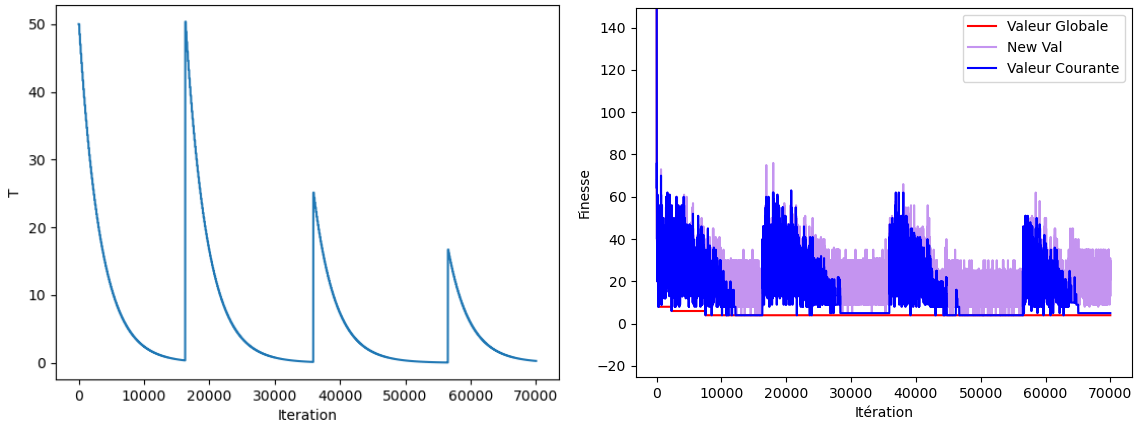
4.1.5 Non alloc and Contact



(a) (b)

FIGURE 5 – Résultat pour la journée du 21 juin 2016 en utilisant l'opérateur NAAC

Au bout de 70000 itérations on obtient 4. Au bout de 50 itérations, on est déjà à 16.



(a) (b)

FIGURE 6 – Résultat pour la journée du 21 juin 2016 en utilisant l'opérateur NAAC en utilisant une autre loi de décroissance de T

4.2 Combinaison d'opérateurs

4.2.1 Non alloc and contact et Mutate

4.2.2 Mutate et Randomize Subset

5 Conclusion

Références

[1] Sang Hyun Kim, Eric Feron, John-Paul Clarke, Aude Marzuoli, and Daniel Delahaye. Airport gate scheduling for passengers, aircraft, and operations. *Journal of Air Transportation*, 25(4) :109–114, 2017.

[2] Chuhang Yu, Dong Zhang, and Henry YK Lau. Mip-based heuristics for solving robust gate assignment problems. *Computers & Industrial Engineering*, 93 :171–191, 2016.

[3] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598) :671–680, 1983.

[4] Daniel Delahaye, Supatcha Chaimatanan, and Marcel Mongeau. Simulated annealing : From basics to applications. *Handbook of metaheuristics*, pages 1–35, 2019.

[5] Ningning Zhao and Mingming Duan. Research on airport multi-objective optimization of stand allocation based on simulated annealing algorithm. *Mathematical Biosciences and Engineering*, 18(6) :8314–8330, 2021.

[6] Soi-Hoi Lam, Jia-Meng Cao, and Henry Fan. Development of an intelligent agent for airport gate assignment. *Journal of Air Transportation*, 7(2), 2002.