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 et 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 que l'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 par des délais courts et de forte densité. Un retard sur un vol pourrait entraîner, 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'entraîner 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 méta-heuristiques, 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, 6]. 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.* [7] 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. Tout en maximisant également le nombre d'avions 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'accueillir 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 les découper en plusieurs opérations. Si le séjour n'est tractable qu'une seule fois, alors les deux opérations qui en découlent ont comme préférence 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 correspondant au milieu de séjour, ne nécessite pas un accès direct au terminal car l'avion n'a pas à être chargé en marchandises ou passagers. Elle peut donc être attribué à un parking dit "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 accrue 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 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 décroît 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 réitère 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 inconvénient du recuit simulé réside dans le choix des paramètres.
La température est intrinsèquement 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 utilisée à chaque itération. La représentation choisie est celle d'un tableau ayant comme indice les différents séjours des avions. 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 élément 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. Le décodage est représenté en Figure 2.

Concernant la vérification des contraintes de planning lors du décodage, lorsque deux séjours sont alloués au même moment sur le même parking, celui ayant le moins de parkings compatibles sera prioritaire, l'autre sera non alloué. Si ces deux séjours ont le même nombre de parkings compatibles, ils auront tous les deux une chance sur deux d'être alloué à ce parking.
Cette partie de décodage est cruciale et est souvent la plus longue à réaliser du fait qu'il faut parcourir toute la solution.

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 nouvelle 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 aléatoirement parmi la liste de ses parkings compatibles. Cet

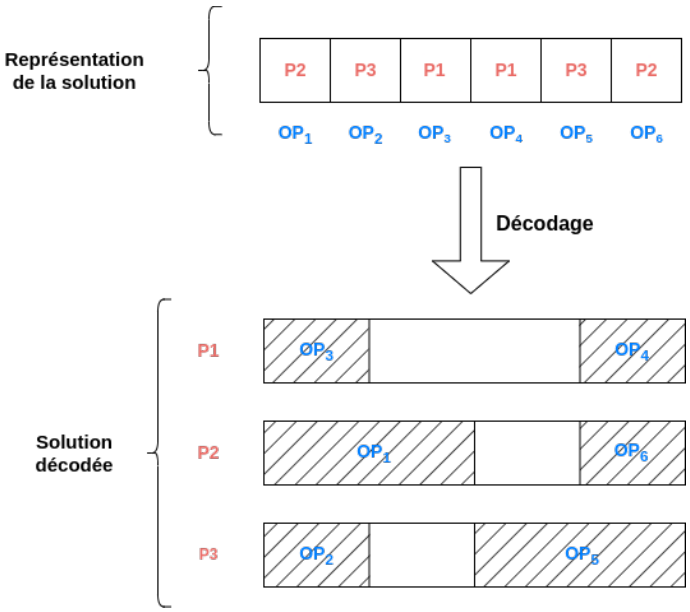


FIGURE 2 – Schéma du décodage

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.

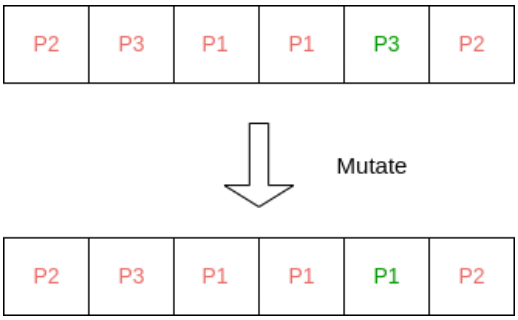


FIGURE 3 – Schéma "Mutate"

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 sera beaucoup plus grand. Ainsi, il est plus facile d'explorer l'ensemble de recherche des solutions. Mais, au contraire, il est plus difficile d'effectuer une recherche locale. Cet opérateur est donc utile pour fortement perturber la solution.

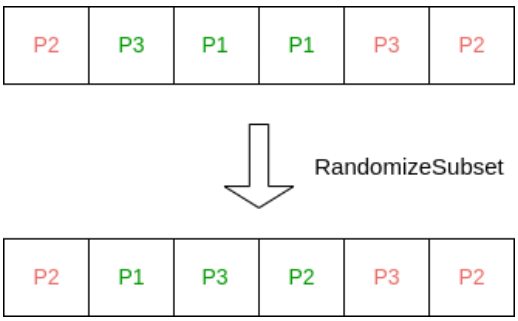


FIGURE 4 – Schéma "Randomize Subset"

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 choisit aléatoirement un parking parmi ces parkings compatibles. En ne changeant que deux valeurs parmi la solution le voisinage d'une solution est donc restreint. En utilisant cet opérateur, la contrainte de compatibilité des parkings n'est pas vérifié et donc peut mener à une recherche plus longue.

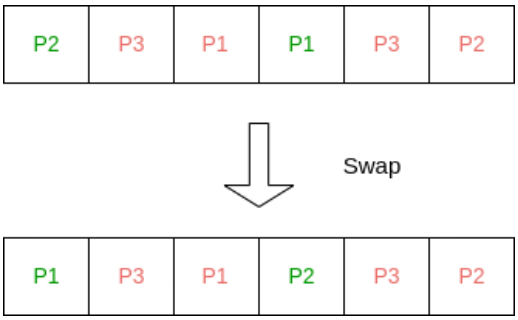


FIGURE 5 – Schéma "Swap"

3.3.4 Non alloc

L'opérateur désigné "Non alloc" a pour but de modifier toutes opérations qui ne sont pas allouées. Il parcourt toute la solution, et lorsque qu'un élément n'a aucun parking attribué – *i.e.* une valeur valant -1, 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. Cet opérateur a donc pour but de "muter" les -1. C'est pourquoi sur les graphiques affichés par la suite, cet opérateur sera appelé MMO, pour "Mutate Minus One".

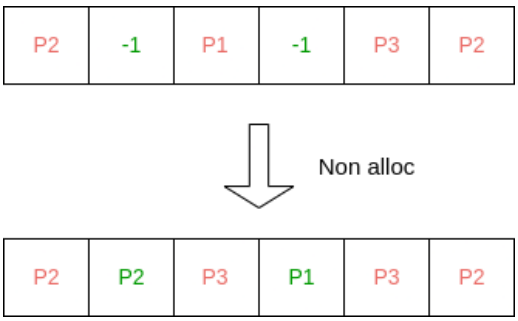


FIGURE 6 – Schéma "Non alloc"

3.3.5 Non alloc and Contact

L'opérateur "Non alloc and Contact" (NAAC) est une amélioration de l'opérateur "Non alloc". Car en plus de modifier les opérations qui ne sont pas allouées. Cet opérateur modifie également les opérations qui sont attribuées sur des parkings "larges" 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

Dans cette partie nous présentons d'abord les résultats du recuit simulé pour une journée puis pour une semaine entière.

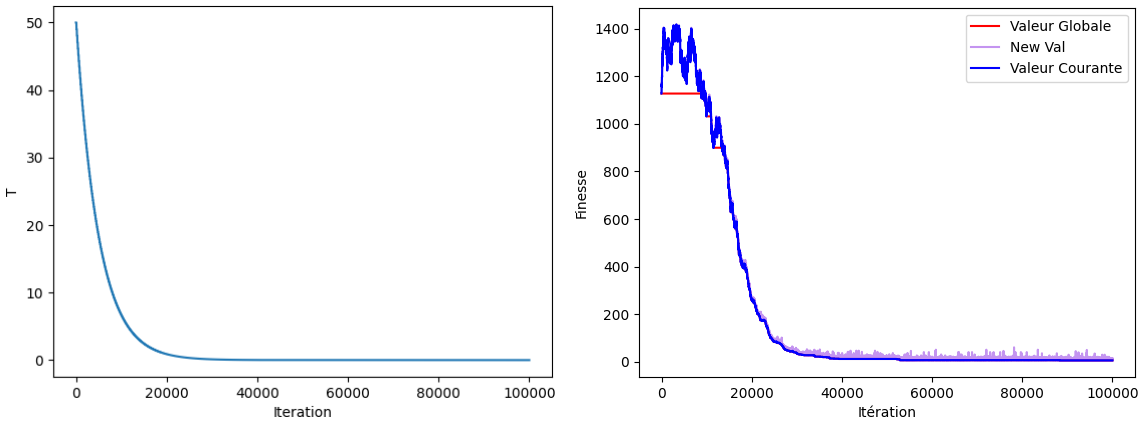
4.1 Résultats pour une journée

Les résultats montrés dans cette partie sont réalisés sur la journée du 26 juin 2016 représentant 373 séjours différents, qui ont été séparés en 578 opérations en considérant les remorquages possibles. Le nombre de parkings disponibles est de 145.

4.1.1 Opérateur

Dans cette partie sont affichés les résultats des différents opérateurs.

4.1.1.1 Mutate



(a) (b)

FIGURE 7 – Résultat pour la journée du 26 juin 2016 en utilisant l'opérateur "Mutate"

La Figure 7a représente la valeur de la température en fonction du nombre d'itérations. La température initiale est fixée à 50, et $\alpha = 0.98$. De plus, lors du recuit simulé, 100 itérations sont calculées pour une même température T. La limite du recuit simulé est fixée à 100 000 itérations.
Avec l'opérateur "Mutate", les résultats obtenus pour la journée du 26 juin 2016 sont montrés Figure 7b. La finesse représentée en ordonnée correspond à la valeur

de la solution par la fonction objectif. Trois différentes valeurs sont représentées. La valeur globale qui correspond à la meilleure solution obtenue jusqu'à cette itération. La valeur courante représente la solution qui est utilisée par les opérateurs pour calculer une nouvelle solution. Et "New Val" correspond à la finesse de cette nouvelle solution.
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. Avec ces paramètres, 66 secondes ont été nécessaires pour compléter l'exécution.

4.1.1.2 Randomize Subset

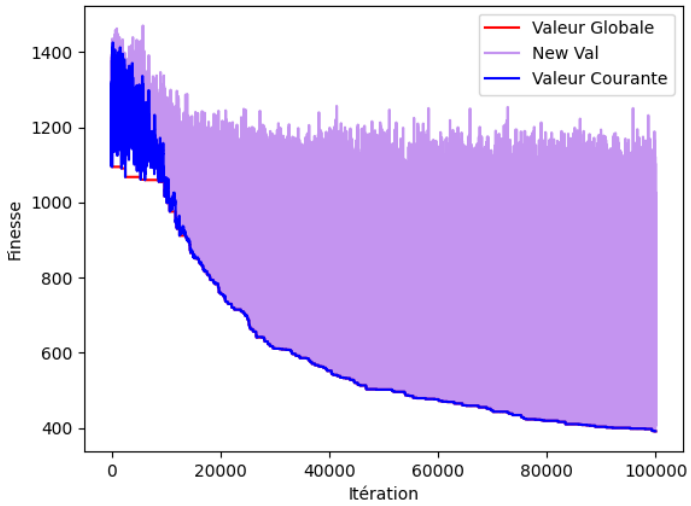


FIGURE 8 – Résultat pour la journée du 26 juin 2016 en utilisant l'opérateur "Randomize Subset"

La Figure 8 représente l'évolution de la valeur globale et courante en utilisant l'opérateur "Randomize Subset". La loi de décroissance de la température est la même que celle montrée Figure 7a. 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. Au bout de 100 000 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 décroître encore même après 100 000 itérations. Il peut donc être tentant de tester ce qu'il en est avec plus d'itérations.

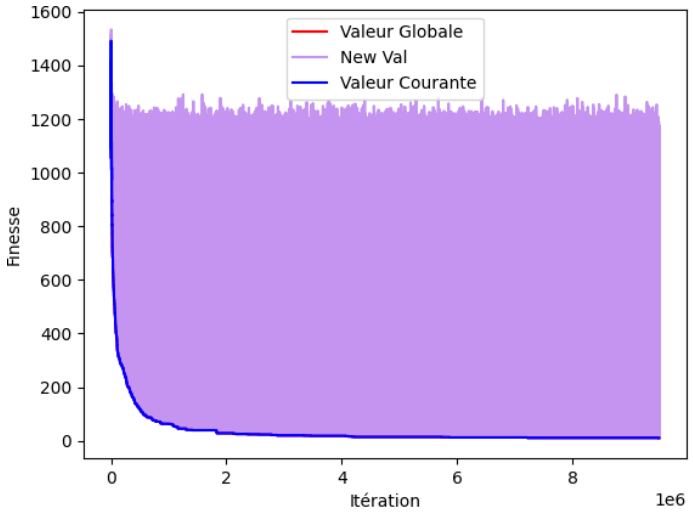


FIGURE 9 – "Randomize Subset" avec beaucoup d'itérations pour la journée du 26 juin 2016

Non sans peine, l'algorithme parvient à une solution plus ou moins correcte au bout de presque 10 millions d'itérations. La solution obtenue après 2 heures et 10 minutes présente une finesse de 10 avec un seul séjour non alloué. On remarque que cette solution est moins bonne que celle trouvé par "Mutate" en seulement 66 secondes. Cela permet de conclure que "Randomize Subset" n'est pas un bon opérateur quand il fonctionne tout seul.

4.1.1.3 Swap

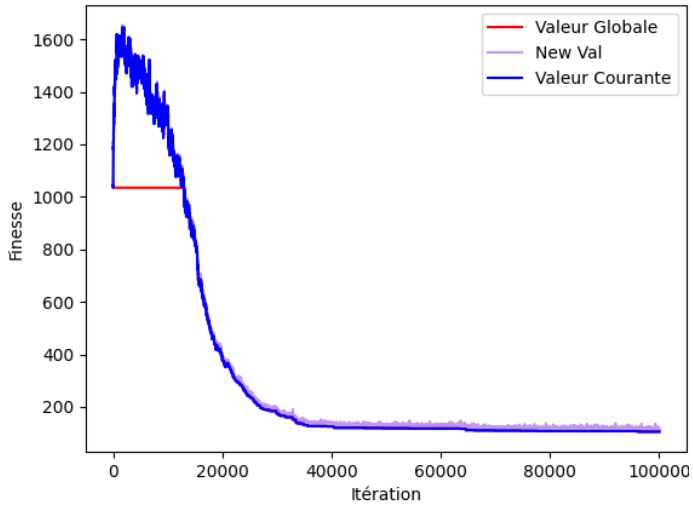
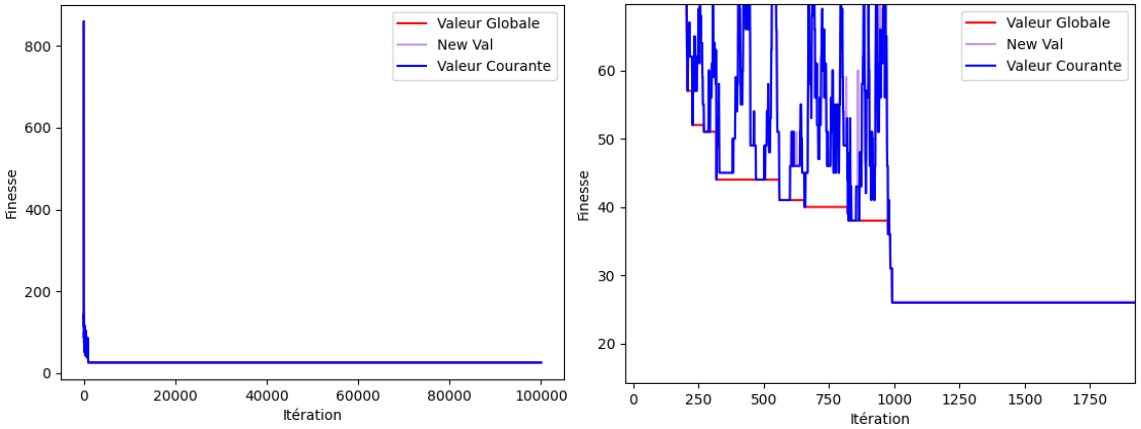


FIGURE 10 – Résultat pour la journée du 26 juin 2016 en utilisant l'opérateur "Swap"

La Figure 10 représente l'évolution de la valeur globale, de la valeur courante ainsi que les nouvelles solutions obtenues à chaque itération en utilisant uniquement l'opérateur "Swap". La loi de décroissance de la température est la même que celle montrée Figure 7a. Cette courbe a le même profil que celle de l'opérateur "Mutate" sur la Figure 7b. 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.

4.1.1.4 Non alloc



(a) (b)

FIGURE 11 – Résultat pour la journée du 26 juin 2016 en utilisant l'opérateur "Non alloc"

La Figure 11 représente les résultats obtenus en utilisant uniquement l'opérateur "Non alloc". La loi de décroissance de la température est la même que celle montrée Figure 7a. Ces résultats nous montrent une convergence presque instantanée vers une solution. On observe qu'en seulement 1 000 itérations, l'algorithme a convergé vers sa solution finale de finesse 26.
L'opérateur "Non alloc" a pour effet d'allouer les séjours non affectés. Mais lorsque toutes les opérations sont allouées, ou lorsqu'il est impossible d'affecter une nouvelle opération sans modification des séjours déjà alloués, cet opérateur ne permet aucune détérioration de la solution et ne peut plus sortir de son optimum local. De plus, cet opérateur parcourt toute la solution encodée à chaque itération et met donc plus de temps que des opérateurs comme "Mutate" ou "Swap". On relève une durée de 95 secondes pour 100 000 itérations.

4.1.1.5 Non alloc and Contact

La Figure 12 représente les résultats obtenus en utilisant uniquement l'opérateur "Non alloc and Contact". La loi de décroissance de la température est la même que celle montrée Figure 7a. Ces résultats nous montrent une convergence rapide vers une solution également. On constate une meilleure performance que l'opérateur "Non alloc". En effet, au bout de 100 itérations, la finesse est déjà de 19. Au bout de 25 000 itérations, la convergence est atteinte avec une finesse de 5. Le temps d'exécution pour 100 000 itérations avec cet opérateur est de 104 secondes. La solution est satisfaisante ; tous les séjours sont alloués. Elle est de plus trouvée très rapidement. Cependant, nous devons tester sa réelle efficacité sur des instances plus compliquées.

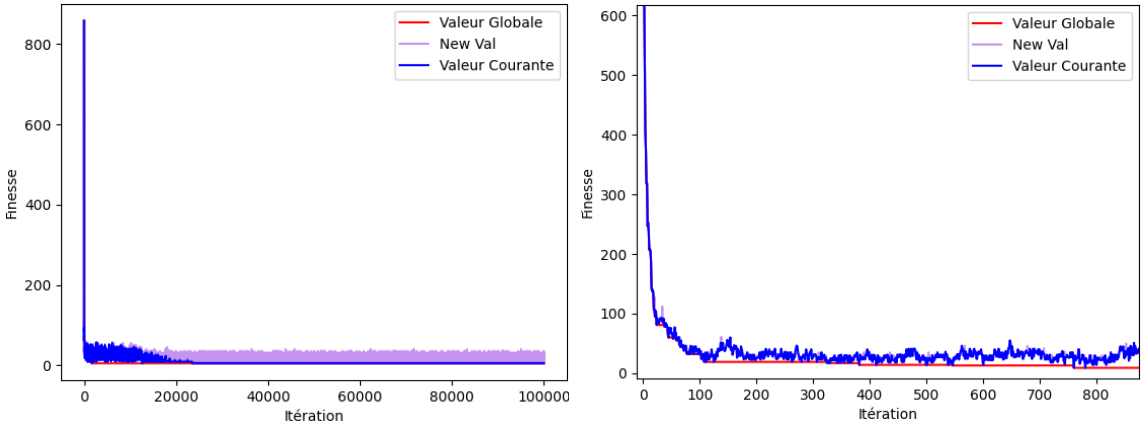


FIGURE 12 – Résultat pour la journée du 26 juin 2016 en utilisant l'opérateur NAAC

4.1.2 Comparaison des simples opérateurs

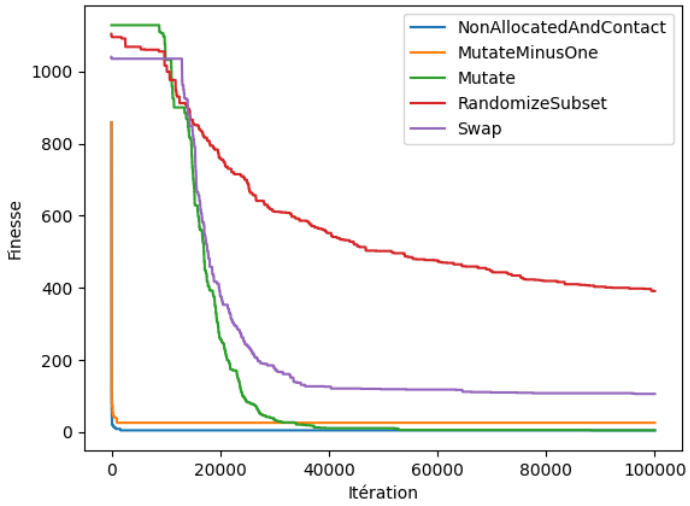


FIGURE 13 – Comparatif des opérateurs pour la journée du 26 juin 2016

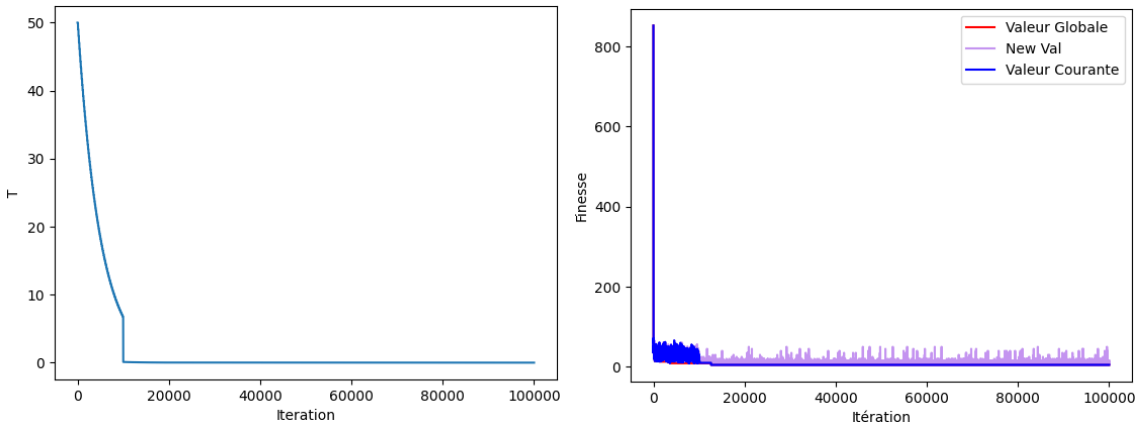
En Figure 13 sont représentées les performances des différentes opérateurs. Cette comparaison met en exergue l'efficacité de l'opérateur NAAC – "Non alloc and Contact" – face aux autres. L'opérateur "Mutate" atteint un optimum de même finesse que celui de NAAC, mais en plus longtemps. Cependant, en raison de sa conception, "Mutate" peut être plus robuste contre des optimum locaux.

4.1.3 Combinaison d'opérateurs

Après avoir observé les performances de chaque opérateur, nous nous penchons désormais sur les performances des combinaisons de deux opérateurs. Combinaisons qui ont pour but d'améliorer l'approximation de l'optimum global.

4.1.3.1 NAAC puis Mutate

Par les résultats précédents, on distingue aisément l'opérateur "Non alloc and Contact" pour sa capacité à converger vers une solution convaincante. Une idée



(a) (b)

FIGURE 14 – Résultat pour la journée du 26 juin 2016 en combinant les opérateurs "Non alloc and Contact" puis "Mutate"

serait alors d'utiliser cet opérateur pendant 10 000 itérations, puis, utiliser un autre opérateur tel que "Mutate" pour essayer d'améliorer encore plus.

Comme nous pouvons le voir sur la Figure 14b, la combinaison des opérateurs "Non alloc and Contact" et "Mutate" permet d'atteindre, en 75 secondes, une solution de finesse 5 qui semble être l'optimum global.
Après 10 000 itérations, l'algorithme est à la finesse 6 et on fixe la température T à 0.5 pour réduire fortement la probabilité d'accepter de moins bonnes solutions par la suite. La loi de décroissance de T est ainsi présenté sur la Figure 14a. Le fait d'avoir ensuite appliquer "Mutate" a permis d'améliorer et de passer à 5.

4.1.3.2 Mutate et Randomize Subset

Nous avons constaté l'efficacité de "Mutate" et son principal défaut réside dans le fait qu'il ne perturbe que peu la solution courante. Combiner cet opérateur avec un autre opérateur qui perturbe fortement la solution tel que "Randomize Subset" peut s'avérer intéressant.

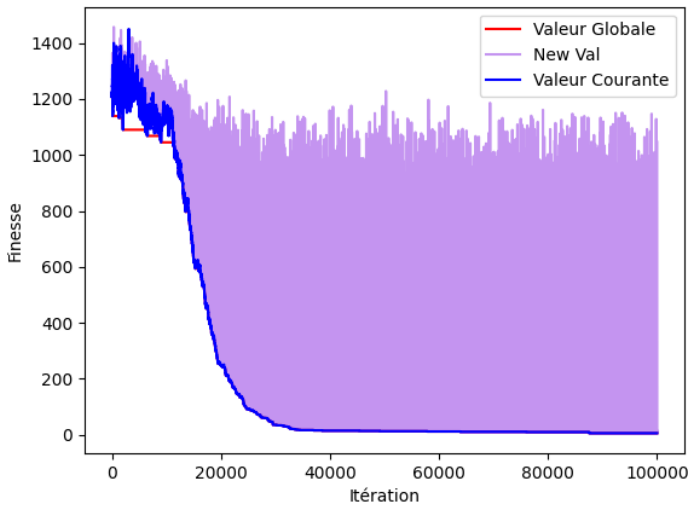
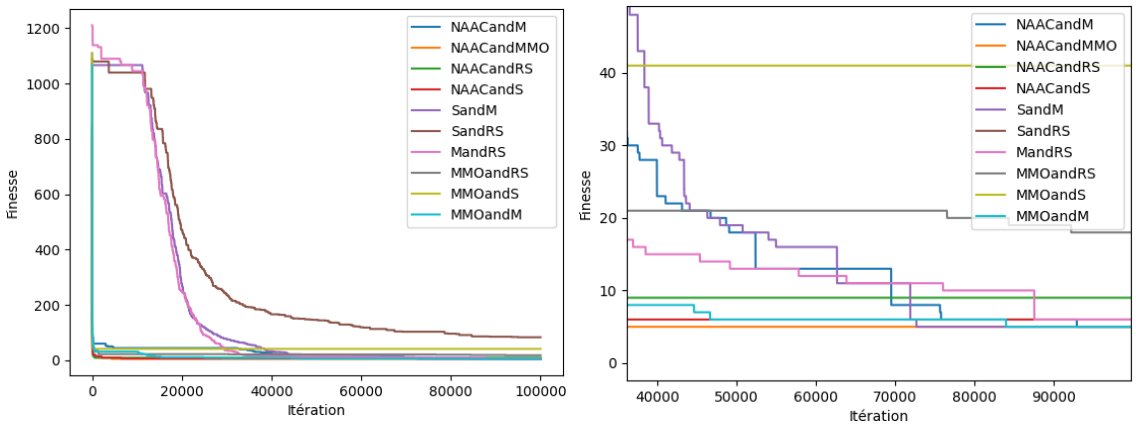


FIGURE 15 – Résultat pour la journée du 26 juin 2016 en combinant les opérateurs "Mutate" et "Randomize Subset"

La Figure 15 représente les résultats obtenus en utilisant l'opérateur "Mutate"

combiné avec l'opérateur "Randomize Subset". La loi de décroissance de la température est la même que celle montrée Figure 7a. En 69 secondes, l'algorithme trouve une solution satisfaisante (tous les séjours sont alloués) de finesse 6. Bien que la finesse résultante est, de très peu, moins bonne que celle obtenue en utilisant uniquement "Mutate", nous pouvons imaginer que cette combinaison d'opérateurs s'avère efficace sur des instances nettement plus compliquées. Le fait d'appliquer "Randomize Subset" permet de sortir des possibles optima locaux trouvés par l'opérateur "Mutate". Cela pourrait être crucial pour des instances plus contraignantes.

4.1.4 Comparaison des combinaisons d'opérateurs



(a) (b)

FIGURE 16 – Comparatif des combinaisons d'opérateurs pour la journée du 26 juin 2016

La Figure 16 représente les résultats des combinaisons d'opérateurs. Sans surprise, les opérateurs qui étaient déjà efficaces lorsqu'ils agissaient seuls le restent en étant combinés à d'autres. D'autre part, les opérateurs moins efficaces seuls, tels que "Randomize Subset" ou encore "Swap", présentent quelques améliorations lorsqu'ils sont combinés à d'autres opérateurs. Par exemple, en 100 000 itérations, "Randomize Subset" seul trouve une solution de finesse 391 (cf. 4.1.1.2) et "Swap" seul trouve une solution de finesse 106 (cf. 4.1.1.3). Lorsque ces deux opérateurs sont combinés, l'algorithme trouve une solution de finesse 83.
La solution obtenue en utilisant l'opérateur "NAAC puis Mutate" sous forme de planche est montrée en Annexe A.

4.1.5 Bilan

Tout ces résultats nous montrent la performance du recuit simulé pour résoudre un problème de GAP sur des instances de la vie réelle. Nous disposons des données de la semaine du 20 juin 2016 au 26 juin 2016 et nous avons testé les performances de chaque opérateur/combinaison de deux opérateurs pour la journée du 26 juin 2016 qui était la plus chargée de la semaine. Cependant, certains opérateurs arrivent très rapidement à un optimum qui semble global, sans avoir même à les combiner entre eux. Discriminer les performances des opérateurs/combinaison de deux opérateurs n'est pas donc pas idéal sur ce genre d'instance. Il est alors nécessaire de compliquer les instances afin d'apprécier les différences entre opérateurs.

4.2 Résultats pour une semaine

Les résultats montrés dans cette partie sont réalisés sur la semaine du 20 au 26 juin 2016 représentant 2305 séjours différents – qui ont été séparés en 3375 opérations. Le nombre de parkings utilisés est toujours de 145.

4.2.1 Opérateur

Comme pour les résultats précédents, nous présentons les résultats selon chaque opérateur. En raisons des performances des opérateurs "Swap" et "Randomize Subset" lorsqu'ils sont utilisés seuls dans les parties précédentes, ces derniers ne sont pas présentés ici. Ils apparaîtront néanmoins dans le comparatif des simples opérateurs de la section 4.2.2.

4.2.1.1 Mutate

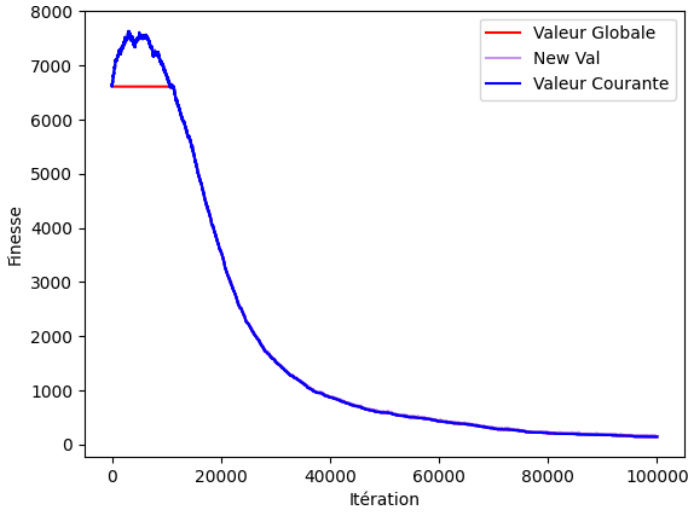


FIGURE 17 – Résultat pour la semaine du 20 au 26 juin 2016 en utilisant l'opérateur "Mutate"

Les résultats de la Figure 17 ont été obtenus avec la loi de décroissance de T décrite en Figure 7a. En 11 minutes et 13 secondes, l'algorithme appliqué avec l'opérateur "Mutate" parvient à une solution de finesse 143 avec 12 séjours non alloués. Bien que cette solution ne soit pas optimale, elle est trouvée de manière assez rapide compte tenu de la grande instance.

On peut observer que la courbe tend à décroître encore au bout de 100 000 itérations. Augmenter le nombre d'itérations peut permettre d'améliorer d'avantage la solution trouvée.

Ainsi, avec 200 000 itérations, on atteint un régime quasi-stationnaire, comme on le remarque sur la Figure 18. Au bout de 25 minutes, la valeur finale est de 75 avec 8 séjours non alloués. On remarque qu'avec deux fois plus d'itérations, la solution obtenue présente une finesse de presque la moitié de la finesse obtenue avec 100 000 itérations.

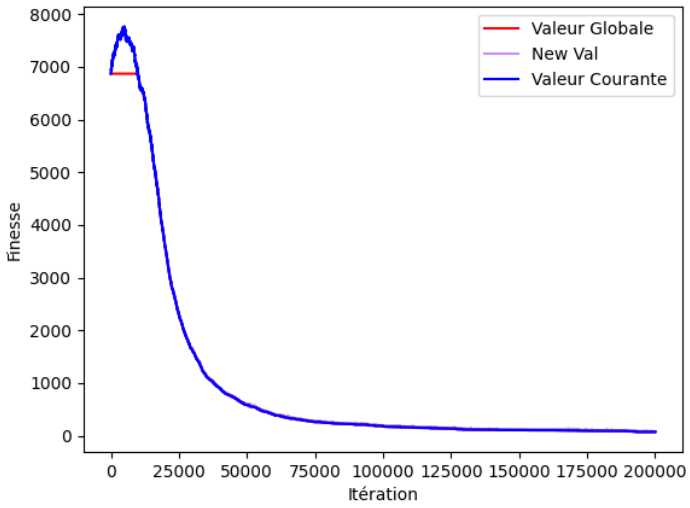
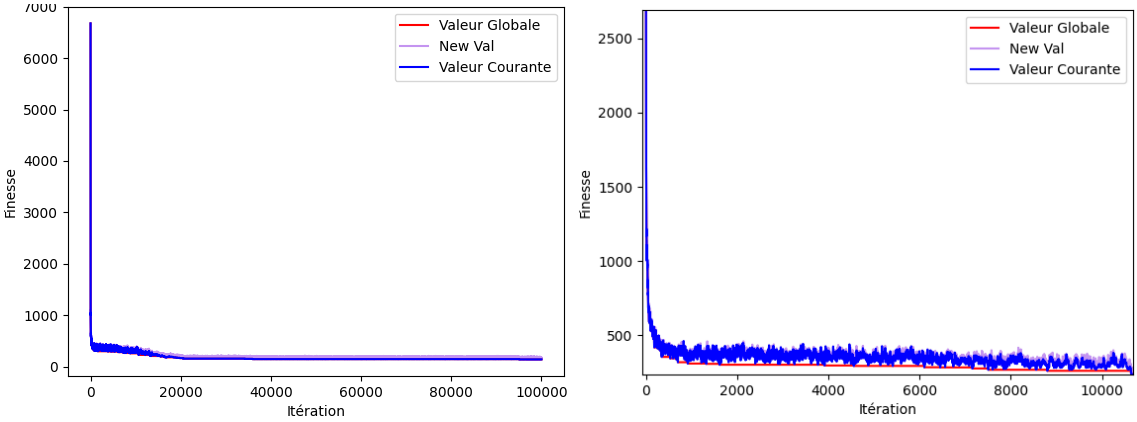


FIGURE 18 – Résultat pour la semaine du 20 au 26 juin 2016 en utilisant l'opérateur "Mutate" sur un plus grand nombre d'itérations

4.2.1.2 Non alloc



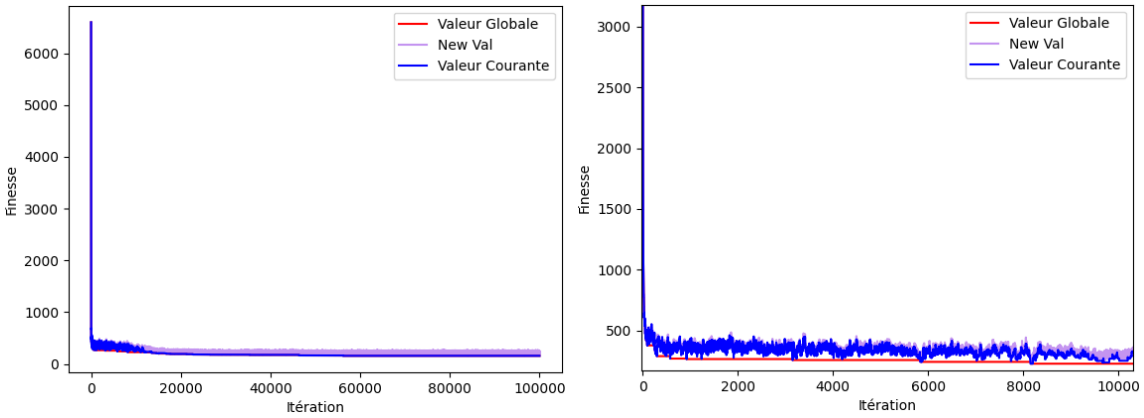
(a) (b)

FIGURE 19 – Résultat pour la semaine du 20 au 26 juin 2016 en utilisant l'opérateur "Non alloc"

Tout comme les résultats de la section 4.1.1.4, les résultats de la Figure 19 nous montrent une forte diminution de la finesse dès les premières itérations. En effet, à l'itération 8 800, nous sommes déjà à une finesse de 259. À partir de 20 000 itérations, l'algorithme ne parvient que très peu à améliorer la solution et parvient, au final, à une solution de finesse 142 au bout de 12 minutes et 17 secondes. Cette solution présente seulement 7 séjours non alloués. Nous constatons que cet opérateur parvient à allouer plus de séjours que l'opérateur "Mutate", mais ne favorise pas le taux de "contact". De plus, cet opérateur a atteint son point de convergence très rapidement et faire plus d'itérations n'améliorerait pas grandement la solution trouvée.

4.2.1.3 Non alloc and Contact

Les résultats de la Figure 20 nous montrent une nouvelle fois l'efficacité de "Non alloc and Contact" pour trouver une solution de finesse basse dès les premières itérations. Nous pouvons constater en Figure 20b qu'à l'itération 8 200, la valeur

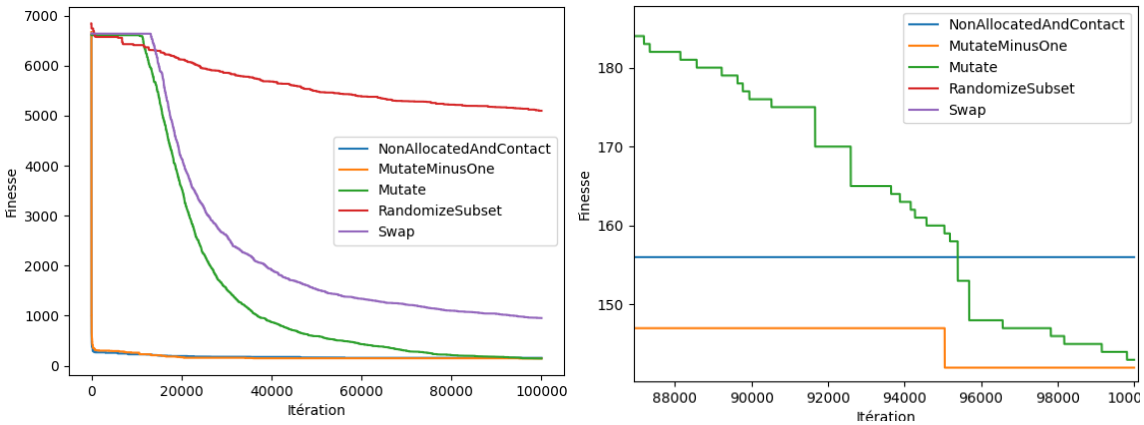


(a) (b)

FIGURE 20 – Résultat pour la semaine du 20 au 26 juin 2016 en utilisant l'opérateur "Non alloc and Contact"

globale vaut déjà 225.
Le recuit simulé a ici duré 15 minutes et 30 secondes pour trouver une solution de finesse 156 avec 28 séjours non alloués. Le "Non alloc and Contact" a convergé sur la Figure 20a et ce depuis longtemps. On s'imagine qu'avec plus d'itérations, il ne parviendra pas à améliorer la présente solution qui est peu convaincante. On commence donc à apercevoir les limites de cet opérateur pour des instances plus compliquées comme celle d'une semaine.

4.2.2 Comparaison des simples opérateurs



(a) (b)

FIGURE 21 – Comparatif des simples d'opérateurs pour la semaine du 20 au 26 juin 2016

La Figure 21a nous montre bien la différence d'attitude pour les opérateurs "Non alloc" et "Non alloc and Contact" par rapport aux autres. Ces derniers, comme énoncé précédemment, atteignent très rapidement leur point de convergence mais ne sont pas évolutifs. On remarque sur la Figure 21b le comportement très prometteur de "Mutate" qui semble pouvoir encore bien améliorer sa solution même après 100 000 itérations. Cette assertion est d'ailleurs prouvé par la Figure 18. Enfin, ce comparatif pointe notamment les limites de l'opérateur "Non alloc and Contact"

utilisé seul. En effet, son utilisation est plus longue (15 min) et la solution obtenue est moins bonne que certains opérateurs plus basiques tels que "Mutate", un opérateur évolutif qui plus est.

4.2.3 Combinaison d'opérateurs

La partie qui suit recense quelques résultats notoires de combinaison d'opérateurs.

4.2.3.1 Mutate et Randomize Subset

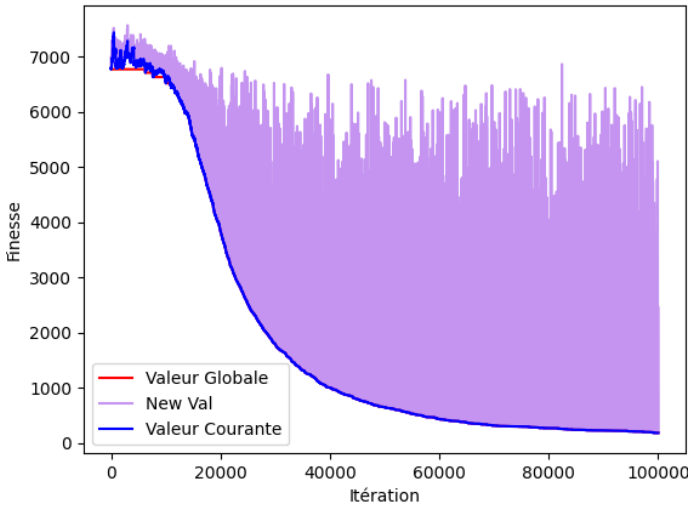


FIGURE 22 – Résultat pour la semaine du 20 au 26 juin 2016 en utilisant les opérateurs "Mutate" et "Randomize Subset"

A la différence de la Figure 15, la Figure 22 n'a pas encore atteint son point de convergence. Cela pourrait être intéressant de voir si on atteint, en un temps raisonnable, une solution plus convenable en laissant tourner l'algorithme plus longtemps. Ici, en 100 000 itérations, la combinaison de ces deux opérateurs parvient à une solution de finesse 187 avec 20 séjours non alloués en 12 minutes et 5 secondes. Ces résultats sont certes moins bons que ceux avec l'opérateur "Mutate" uniquement, mais semblent être grandement améliorables.

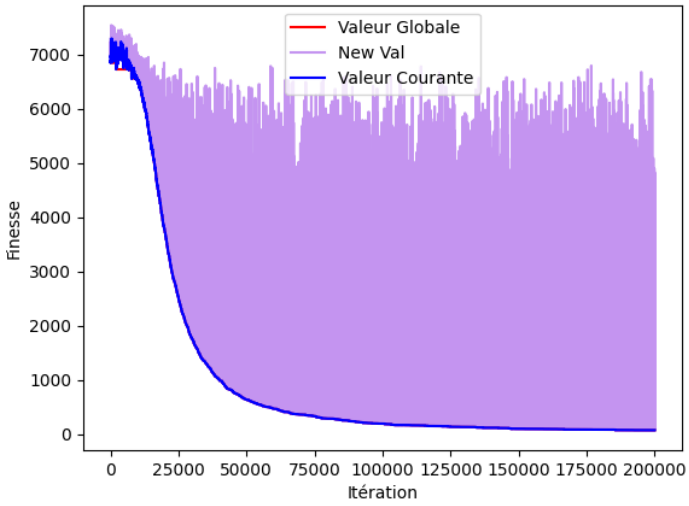
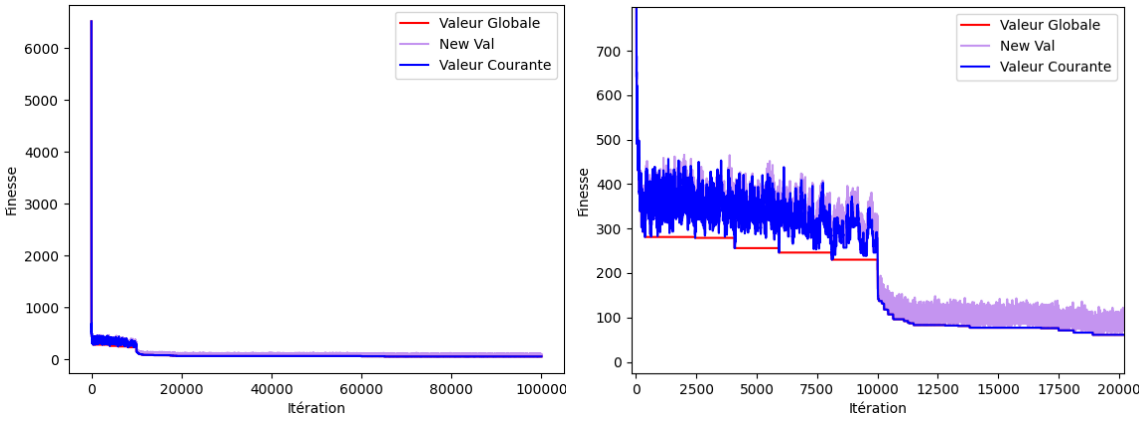


FIGURE 23 – Résultat pour la semaine du 20 au 26 juin 2016 en utilisant les opérateurs "Mutate" et "Randomize Subset" sur un plus grand nombre d'itérations

La Figure 23 nous montre le caractère évolutif de la combinaison entre "Mutate" et "Randomize Subset". En augmentant le nombre d'itérations de 100 000, on obtient une solution de finesse 70 avec 7 séjours non alloués en 24 minutes. On observe un léger contraste d'évolution entre l'opérateur "Mutate" et la combinaison "Mutate" avec "Randomize Subset". En augmentant de 100 000 le nombre d'itérations, l'opérateur "Mutate" est passé d'une finesse de 143 à 75 tandis que la combinaison "Mutate" et "Randomize Subset" est passée de 187 à 70. De plus, la courbe montre qu'il y a encore quelques améliorations possibles sur encore plus d'itérations.

4.2.3.2 NAAC puis MMO



(a) (b)

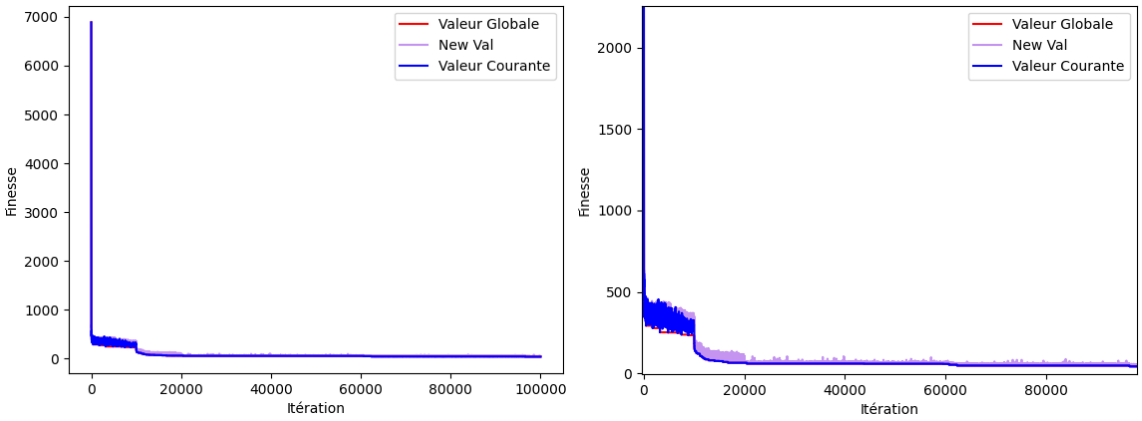
FIGURE 24 – Résultat pour la semaine du 20 au 26 juin 2016 en utilisant les opérateurs "Non alloc and Contact" puis "Non alloc"

Nous avons vu que ces deux opérateurs convergeaient assez vite vers une solution de basse finesse dès les premières itérations. Il peut être intéressant de les combiner. Les résultats de la Figure 24 sont obtenus en utilisant "Non alloc and Contact" pendant 10 000 itérations puis "Non alloc". On remarque que l'opérateur "Non alloc" parvient à améliorer davantage la solution vers laquelle "Non alloc and Contact" convergeait. En revanche, on a ici atteint très rapidement le point de convergence. A

l'itération 20 000, nous sommes à la finesse 64. On obtient finalement une solution de finesse 51 avec seulement 4 séjours non alloués en 14 minutes et 16 secondes.

4.2.3.3 NAAC puis MMO puis Mutate

Dans la partie précédente, nous avons observé des bons résultats pour la combinaison "Non alloc and Contact" et "Non alloc". Ces résultats avaient néanmoins un défaut : celui de ne pas être évolutif. En effet, il n'est pas nécessaire de faire 100 000 itérations avec cette combinaison et on peut imaginer que seul 20 000 itérations suffisent (d'après la Figure 24a). Dans cette partie est donc testée la combinaison de : 10 000 itérations de "Non alloc and Contact" puis 10 000 itérations de "Non alloc" puis 80 000 itérations de "Mutate".



(a) (b)

FIGURE 25 – Résultat pour la semaine du 20 au 26 juin 2016 en utilisant les opérateurs "Non alloc and Contact" et "Non alloc" et "Mutate"

Les résultats de la Figure 25 montrent une légère amélioration par rapport aux résultats de la Figure 24. L'utilisation de "Mutate" n'arrive que très peu à faire mieux que la solution trouvée au bout de 20 000 itérations. Nous parvenons tout de même à une solution de finesse 42 avec seulement 3 séjours non alloués.

4.2.4 Comparaison des combinaisons d'opérateurs

Opérateur	Valeur finale	Séjours non alloués
Mutate	143	12
Randomize Subset	5099	901
Swap	952	87
MMO	142	7
NAAC	156	28
Mutate et Randomize Subset	187	20
MMO puis Mutate	65	8
NAAC puis Mutate	68	8
NAAC puis MMO	51	4
NAAC puis MMO puis Mutate	42	3
MMO puis Mutate et Randomize Subset	78	8

TABLE 1 – Résultats sur une semaine sur 100 000 itérations

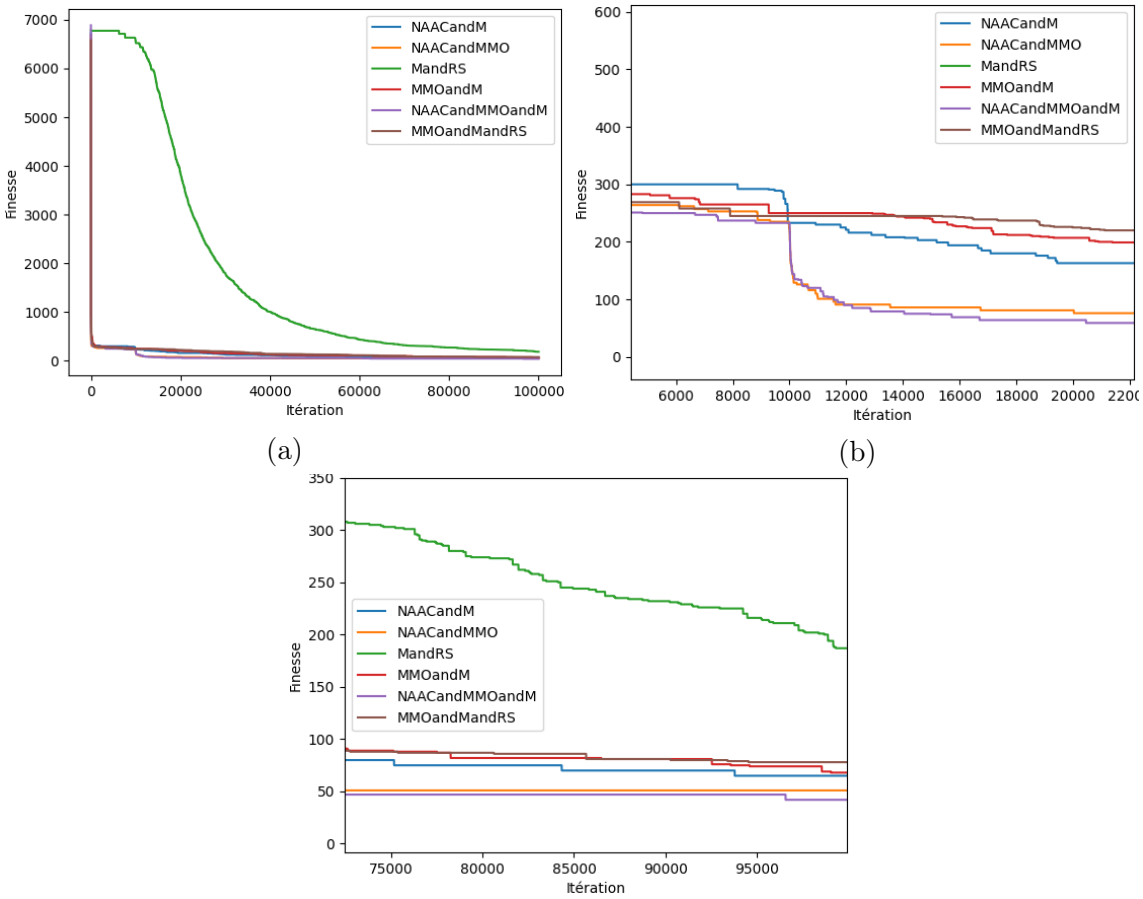


FIGURE 26 – Comparatif des combinaisons d'opérateurs pour la semaine du 20 au 26 juin 2016

La Figure 26 expose les combinaisons d'opérateurs testés les plus efficaces pour résoudre le problème de GAP pour une semaine. Ces combinaisons, mis à part celle utilisant "Mutate" et "Randomize Subset", semblent, au bout de 100 000 itérations, avoir convergé vers une solution globalement assez bonne. La meilleure solution est celle trouvée par les opérateurs "Non alloc and Contact" puis "Non alloc" puis "Mutate" avec une finesse de 42 et 3 séjours non alloués. Cette combinaison n'est, en revanche, pas évolutive contrairement à la combinaison de "Mutate" et "Randomize Subset". Cependant, cette dernière combinaison ne parvient pas à une bonne solution au bout 100 000 itérations et 12 minutes.

4.2.5 Bilan

Bien que, sur toutes instances réelles d'une journée, le recuit simulé avec le ou les bon(s) opérateur(s) parvient à une solution très satisfaisante, il ne permet pas de trouver des solutions sans séjours non alloués. L'algorithme trouve donc des solutions de finesse beaucoup plus élevée. Le temps pris par le recuit simulé pour résoudre l'instance réelle d'une semaine varie entre 11 et 15 minutes. Pour résoudre le problème de GAP de manière plus satisfaisante, il faudrait beaucoup plus de temps. Cela nous est d'ailleurs prouvée par la Figure 23. Enfin, on peut observer que les opérateurs efficaces pour résoudre une instance d'une journée ne le sont pas forcément pour une instance d'une semaine.

4.3 Résultats pour une journée en contraignant le problème

Dans cette partie, certains parkings ont été supprimés. Ainsi, le problème est plus difficile à résoudre et permet donc de différencier plus facilement la performance des opérateurs employés. Au total, ce sont seize parkings qui ont été enlevés, en équilibrant les types avions compatibles. Ainsi, au lieu d'avoir le choix parmi 145 parkings, le problème est restreint à n'utiliser que 129 parkings dans cette partie comme dans la partie suivante. Les résultats de cette partie sont obtenus sur la journée du 26 juin 2016 tandis que dans la partie suivante les résultats seront sur une semaine.

La température initiale est de 50, avec la loi de décroissance décrite à la Figure 7a. Le nombre d'itérations est de 100 000. À noter, que les temps d'exécution des opérateurs sont sensiblement les mêmes et varient entre une et deux minutes en fonction de la complexité des opérateurs. Les résultats des opérateurs sont résumés dans la Table 2.

On observe que le meilleur opérateur, utilisé seul, est Mutate. Il permet d'obtenir une finesse de 15 en allouant tous les avions. On remarque également une nette différence entre MMO ("Non alloc") et NAAC ("Non alloc and Contact"). "Non alloc" a pour effet de minimiser le nombre de séjours non alloués, cependant beaucoup de séjours sont affectés à des parkings "larges". "Non alloc and Contact", au contraire, favorise le nombre de séjours affectés à des parkings "contacts", mais n'arrive pas à allouer tous les avions.

En combinant certains opérateurs, dans l'espoir d'améliorer la solution, en utilisant les forces et faiblesses des différents opérateurs, les résultats obtenus sont intéressants car la finesse est faible et très peu de séjours ne sont pas affectés. Cependant, l'opérateur "Mutate", est l'opérateur qui obtient le meilleur résultat.

Opérateur	Valeur finale	Séjours non alloués
Mutate	15	0
Randomize Subset	456	74
Swap	107	6
MMO	42	1
NAAC	57	9
Mutate et Randomize Subset	20	1
Mutate et Randomize Subset et MMO	17	0
MMO puis Mutate	22	1
NAAC puis Mutate	19	1
NAAC puis MMO	38	2
NAAC puis MMO puis Mutate	26	2
MMO puis Mutate et MMO	16	0
MMO puis Mutate et Randomize Subset	25	2

Table 2 – Résultats sur une journée avec moins de parkings

Les combinaisons utilisées dans la Table 2 sont de trois types différents. Dans le premier cas, le but est de combiner des opérateurs de recherche locale et globale, comme "Mutate et Randomize Subset" et "Mutate et Randomize Subset et MMO". Avec ces deux combinaisons, "Mutate" permet de faire une recherche locale tandis

que "Randomize Subset" recherche dans un voisinage beaucoup plus grand. En ajoutant MMO en plus, cela permet de guider l'algorithme, en le forçant à affecter les séjours. Le deuxième cas correspond à commencer le recuit simulé en utilisant seulement l'opérateur MMO ou NAAC. Cela a pour but de plonger rapidement dans une bonne solution. Ensuite, utiliser des opérateurs tels que "Mutate" ou "Randomize Subset" pour chercher dans le voisinage de cette solution. Le principal inconvénient de ce type de combinaison est qu'il est très probable de rester bloquer dans un optimum local.
Les résultats sont assez similaires entre toutes les combinaisons, on peut supposer que cela est dû à l'opérateur "Mutate" qui obtenait de très bons résultats seul. Le troisième cas consiste à combiner MMO et NAAC, car MMO minimise le nombre de séjours non alloués et NAAC maximise l'attribution aux parkings "contacts". Cependant, le résultat de cette combinaison n'est pas très satisfaisant pour une journée. En revanche, en ajoutant l'opérateur "Mutate", après avoir utilisé NAAC puis MMO, la valeur finale est grandement améliorée.

Les courbes obtenues pour ces différents opérateurs sont affichées Figure 27.

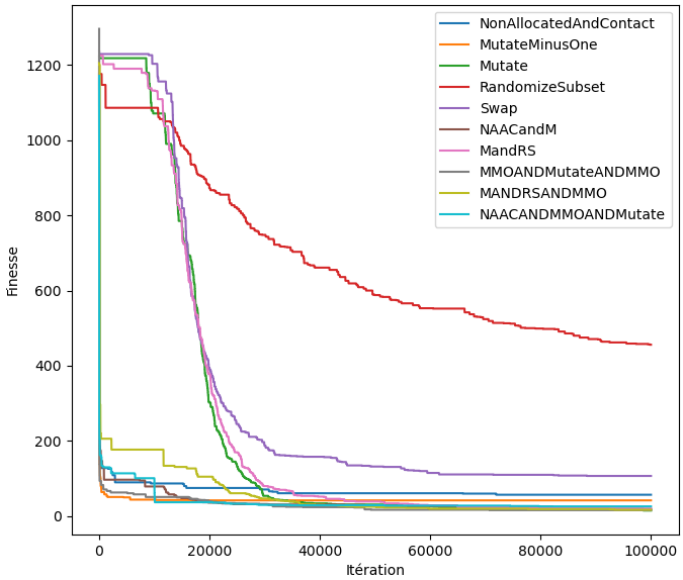


FIGURE 27 – Résultats pour la journée du 26 juin 2016 avec moins de parkings disponibles

La meilleure solution pour cette journée est obtenue avec l'opérateur Mutate, qui obtient une solution finale de finesse 15 en affectant toutes les opérations. Le décodage de cette solution en un planning des parkings est montrée en Annexe B.

4.4 Résultats pour une semaine en contraignant le problème

Dans le but de tester l'efficacité des opérateurs, on applique dans cette partie le recuit simulé sur les données d'une semaine entière. Les paramètres sont définis comme dans les parties précédentes, une température initiale de 50 et un nombre d'itérations limité à 100 000. Les temps d'exécution, en fonction des opérateurs utilisés, varient entre 11 et 16 minutes.

Les résultats obtenus sont détaillés Table 3. On constate que les opérateurs, quand utilisés seuls, ont des performances similaires entre eux en comparant les résultats sur une journée. "Mutate" reste le meilleur opérateur, suivi par MMO.

Néanmoins, en les combinant, on arrive à obtenir de meilleurs résultats que "Mutate". Pour ce faire, les combinaisons ayants réduit la valeur finale, ont comme point commun d'utiliser un opérateur permettant de plonger rapidement dans une bonne solution, *i.e.* MMO ou NAAC. Avec ces opérateurs, une bonne solution est très rapidement trouvée, comme on le voit sur la Figure 28. Ce type de combinaison a pour défaut d'avoir de grandes chances de rester bloquer dans un optimum local. Les combinaisons utilisant les opérateurs classiques comme "Mutate" et "Randomize Subset" permettent également d'obtenir de bons résultats mais nécessitent plus d'itérations pour converger, du fait de la taille du problème.

Opérateur	Valeur finale	Séjours non alloués
Mutate	352	36
Randomize Subset	5524	976
Swap	1219	136
MMO	413	43
NAAC	749	139
Mutate et Randomize Subset	390	40
Mutate et Randomize Subset et MMO	306	28
MMO puis Mutate	211	17
NAAC puis Mutate	237	25
NAAC puis MMO	326	39
NAAC puis MMO puis Mutate	209	20
MMO puis Mutate et MMO	259	26
MMO puis Mutate et Randomize Subset	243	23

TABLE 3 – Résultats sur une semaine avec moins de parkings

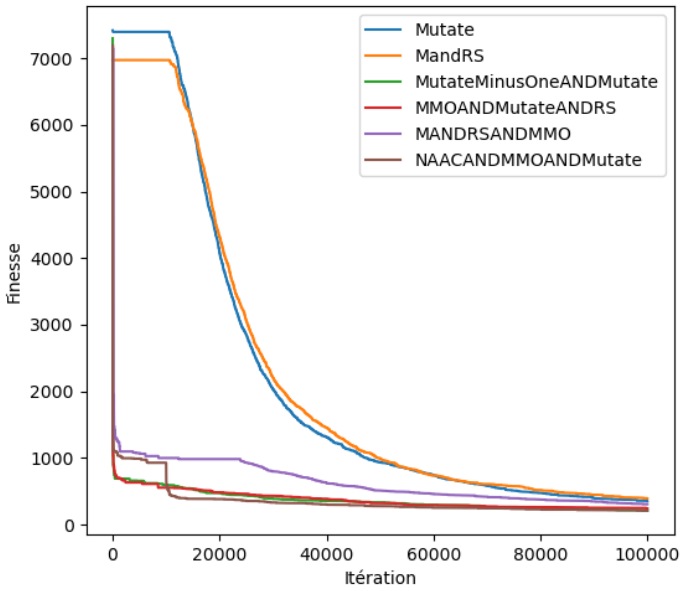


FIGURE 28 – Résultats pour une semaine avec moins de parkings disponibles

Dans la Table 4 et la Figure 29 sont affichés les résultats de trois opérateurs avec 300 000 itérations. Les trois opérateurs convergent vers une solution satisfaisante. "Mutate", permet d'obtenir une finesse de 148 avec 9 séjours non alloués. Et la combinaison "Mutate et Randomize Subset et MMO" atteint également une valeur

finale de 145 avec 10 séjours non alloués. Donc "Mutate" permet d'obtenir de très bons résultats, mais pour ce faire il est nécessaire d'augmenter considérablement le nombre d'itérations. Ainsi, dans l'espoir d'avoir la meilleure solution possible avec un temps d'exécution très long, il est préférable d'utiliser "Mutate". Cependant, si le but est de trouver une bonne solution le plus rapidement possible, il est nécessaire d'utiliser des combinaisons en commençant par les opérateurs plongeant rapidement dans une bonne solution, tels que MMO et NAAC.

Opérateur	Valeur finale	Séjours non alloués
Mutate	148	9
Mutate et Randomize Subset	180	15
Mutate et Randomize Subset et MMO	164	12
MMO puis Mutate	145	10
NAAC puis MMO puis Mutate	162	15

TABLE 4 – Résultats sur une semaine avec moins de parkings avec 300 000 itérations

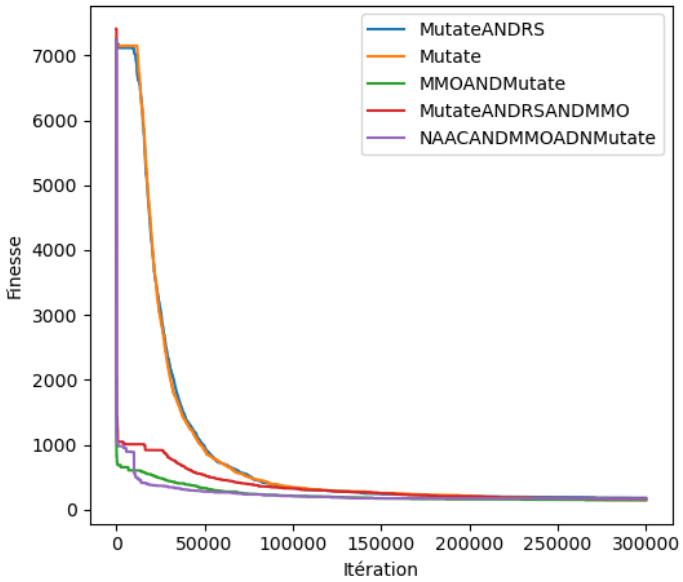


FIGURE 29 – Résultats sur une semaine avec 300 000 itérations

5 Conclusion

Nous avons constaté dans le présent projet l'efficacité du recuit simulé pour résoudre des problèmes tel que le GAP. Le choix du ou des opérateurs à chaque itération est fondamental étant donné les changements drastiques de performance entre certains opérateurs. Nous avons vu que ce choix n'était pas aisé car il peut dépendre de l'instance à traiter. En effet, comme nous l'avons vu, un opérateur ou une combinaison d'opérateurs peut être très efficace sur un type d'instance, mais peu performante sur un autre type d'instance.
Des opérateurs classiques, effectuant une recherche sur des voisinages différents ont été comparés à des opérateurs plus spécifiques à ce problème permettant de guider rapidement vers une bonne solution, avec comme risque de rester bloquer dans un optimum local. Ces opérateurs spécifiques à ce problème permettent dans la majorité des cas d'obtenir une solution convenable en un nombre limité d'itérations. Cependant, les opérateurs classiques permettent, en un grand nombre d'itérations, d'obtenir de très bonnes solutions, avec un risque plus faible de rester bloquer dans un optimum local.

Il existe de nombreux paramètres au recuit simulé, ce qui en fait une des principales difficultés. En plus du choix des opérateurs à traiter, il faut aussi considérer la température initiale, sa loi de décroissance et la proportion d'utilisation de chaque opérateur lorsqu'on décide d'effectuer une combinaison d'opérateurs. Il faut aussi paramétrer les pénalités dans la fonction objectif. Tester toutes les possibilités de paramétrage est donc une tâche ardue et seules quelques possibilités ont été traitées ici.

6 Ouvertures

Le problème de GAP considère ici certaines contraintes. Dans la réalité, d'autres contraintes peuvent venir s'additionner en fonction des préférences de la compagnie exploitante de l'aéroport. Il peut y avoir notamment la contrainte d'ombrage qui stipule qu'il ne peut y avoir deux avions au même moment sur deux parkings adjacents. Le nombre de tractage n'a pas non plus été considéré ici. Minimiser ce nombre peut être une préférence de la compagnie exploitante. Intégrer ces nouvelles contraintes et préférences au recuit simulé pourrait grandement affecter le temps de recherche d'une bonne solution. De plus, une autre représentation de la solution peut être envisagée, en représentant en indice les parkings avec comme éléments les séjours affectés sur ce parking. Cette nouvelle modélisation ayant pour but de faciliter le processus de décodage et ainsi améliorer la performance générale du recuit simulé.

Annexe A

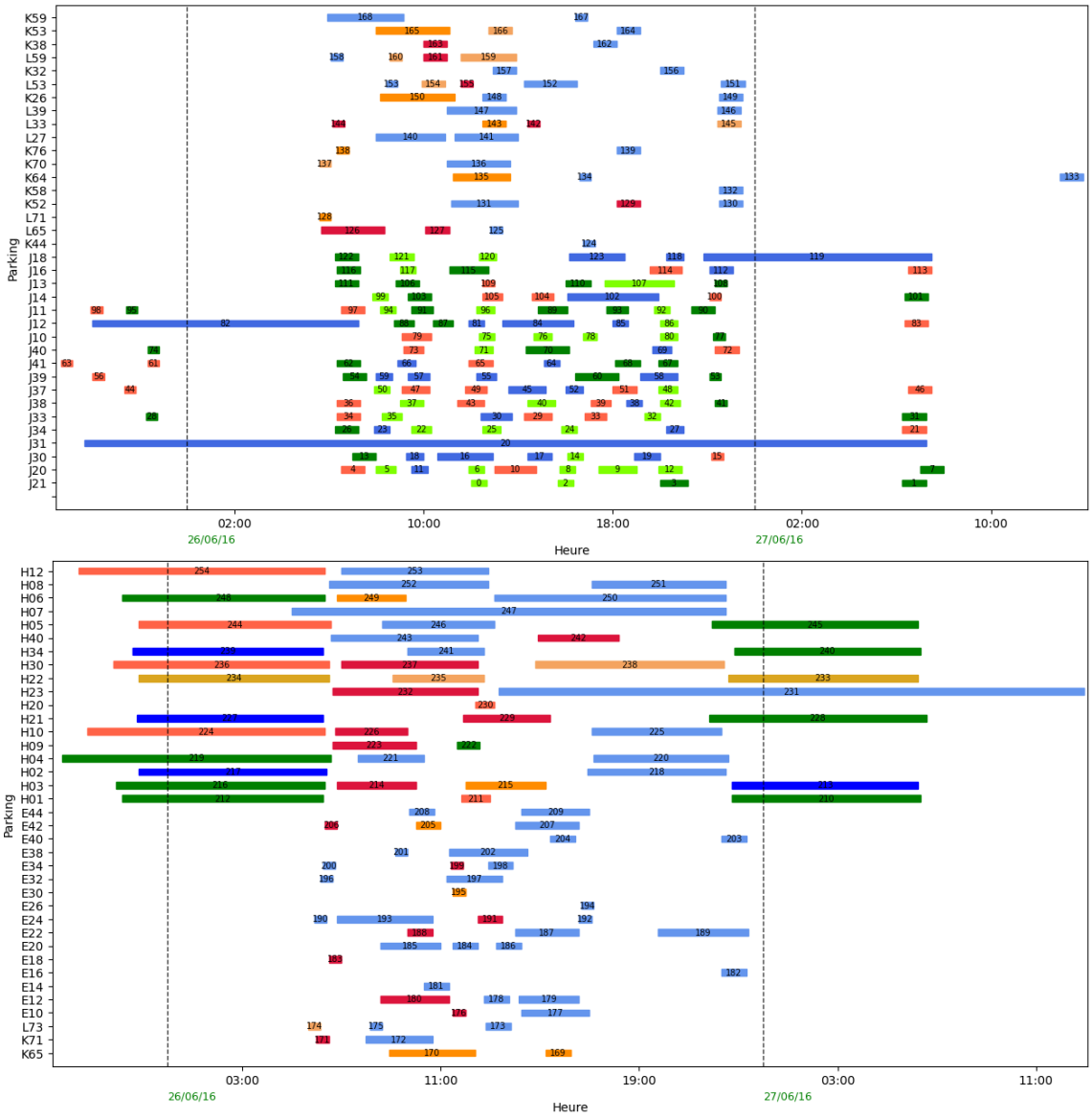


FIGURE 30 – Planche pour la journée du 26 juin 2016, avec 145 parkings disponibles en utilisant l'opérateur "NAAC puis Mutate"

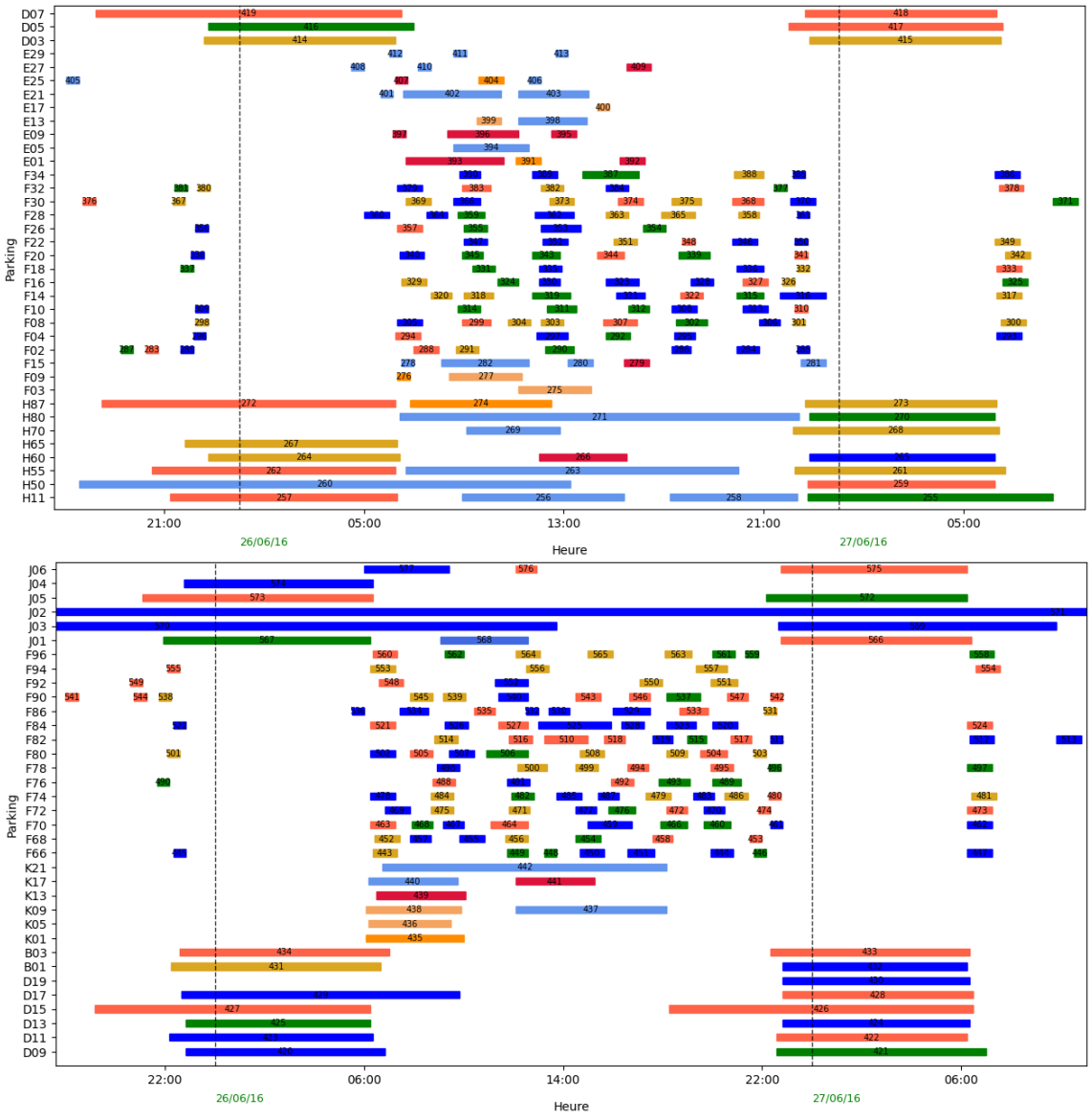


FIGURE 30 – Planche pour la journée du 26 juin 2016, avec 145 parkings disponibles en utilisant l'opérateur "NAAC puis Mutate"

Annexe B

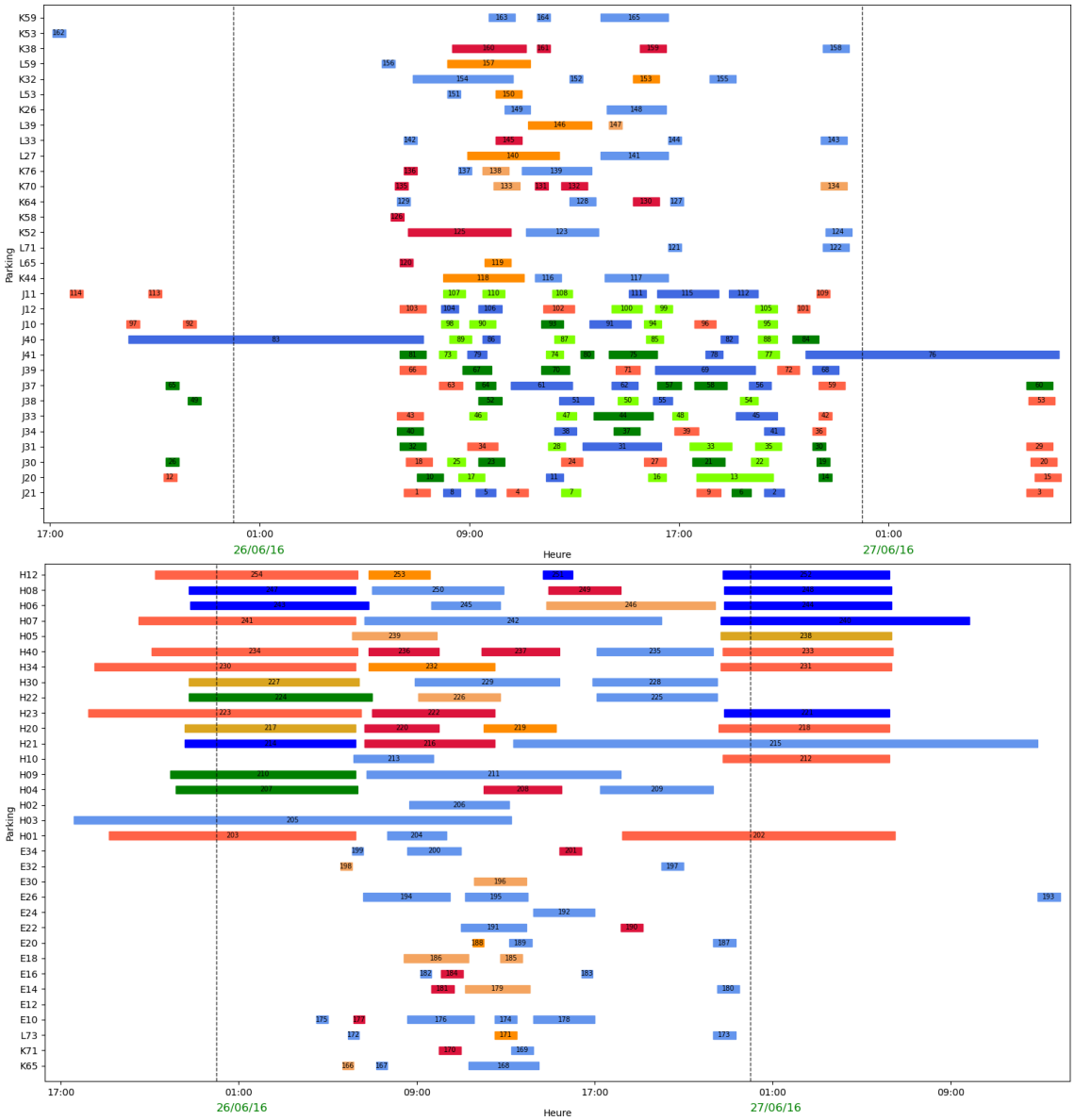


FIGURE 31 – Planche pour la journée du 26 juin 2016, avec 129 parkings disponibles en utilisant l'opérateur Mutate

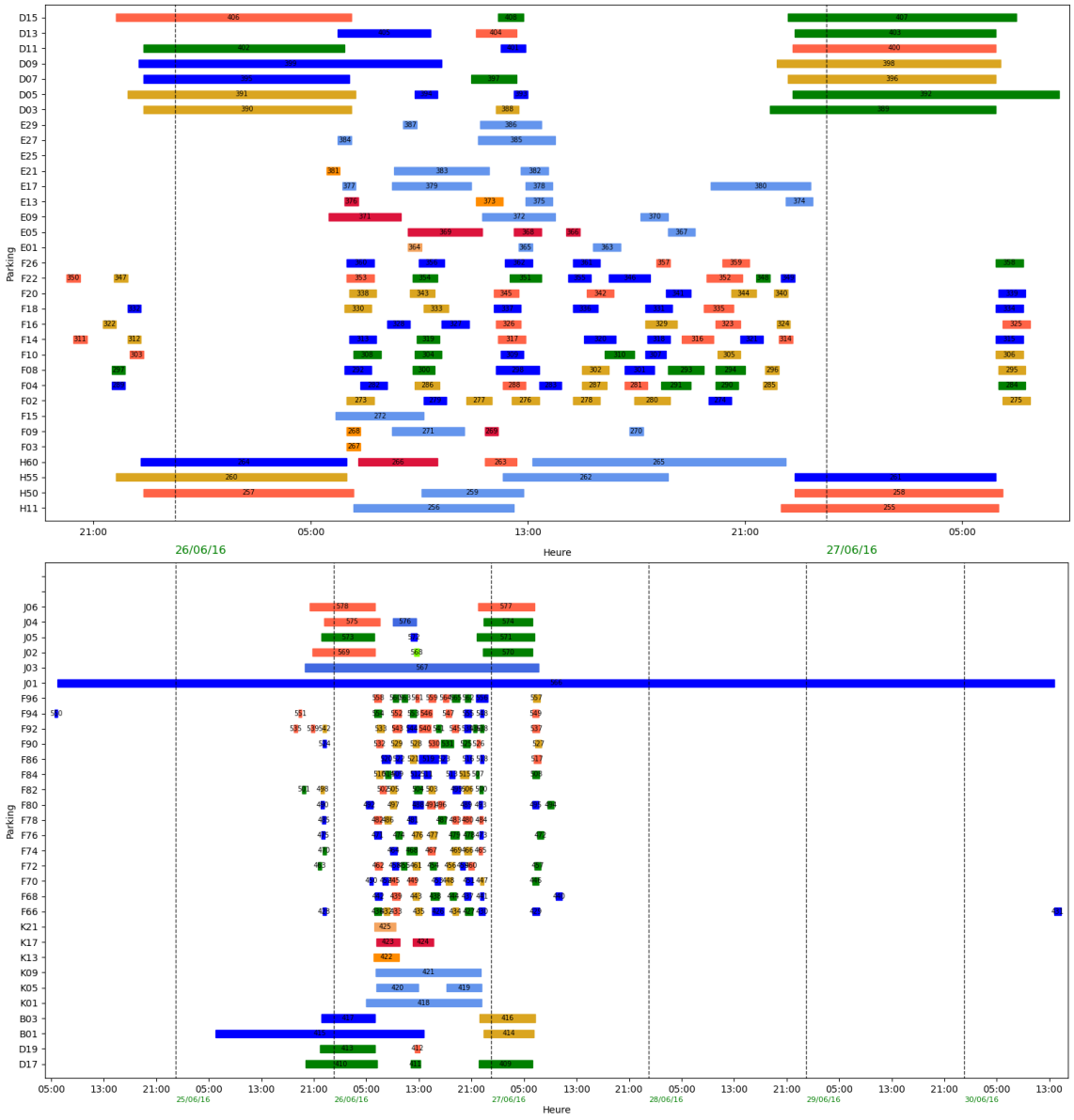


FIGURE 31 – Planche pour la journée du 26 juin 2016, avec 129 parkings disponibles en utilisant l'opérateur Mutate

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] Jiaming Su, Minghua Hu, Yingli Liu, and Jianan Yin. A large neighborhood search algorithm with simulated annealing and time decomposition strategy for the aircraft runway scheduling problem. *Aerospace*, 10(2) :177, 2023.

[7] 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.