Sorbonne Université (Master ANDROIDE)

Projet MAOA
Tournées de techniciens

Auteurs :
Clément Boisson
Binh Thanh Luong

Encadrant :
Pierre Fouilhoux

25 mars 2019

Table des matières

1 Introduction

Dans le cadre de l'UE MAOA du master Androide, nous avons effectué un projet. Ce projet est proposé par M. Fouilhoux. Il consiste en la résolution d'un problème de tournées de techniciens de manière approchée et de manière exacte. Puis nous devions ajouter des contraintes rendant le problème plus difficile à résoudre.

Une instance du problème de tournées des techniciens est un ensemble de points, parmi lesquels on trouve un entrepôt et des clients. Chaque client a une demande à satisfaire. Des véhicules vont donc partir de l'entrepôt et livrer la demande de chacun des clients puis retourner à l'entrepôt. Les véhicules ont une capacité limitée, donc il faut que la somme des demandes d'une tournée ne dépasse pas la capacité maximale des véhicules. Le but de ce problème est donc de trouver la tournée de chacun des véhicules de manière à ce que l'ensemble des tournées prennent le moins de temps possibles (il faut minimiser la somme des distances sur chaque tournée).

Ce problème réunit deux problèmes connus, le problème de sac-à-dos multiple (*Bin Packing Problem*) et le problème du voyageur de commerce (*Travelling Salesman Problem*). Ces deux problèmes étant NP-difficiles, le problème de tournées des techniciens est au moins autant difficile à résoudre.

Dans ce rapport, nous expliquons dans un premier temps nos méthodes de résolution pour obtenir de bonnes solutions et des solutions exactes. Puis nous expliquons ensuite comment résoudre le problème si on ajoute de nouvelles contraintes.

2 Première partie

2.1 Résolution approchée

Nous voyons dans cette partie la résolution des instances de ce problème par des heuristiques. Le but de ces heuristiques est d'obtenir de bonnes solutions proches de la solution optimale et de manière rapide. Ce cas de recherche de solutions s'applique si l'entreprise veut une solution dans l'immédiat. Pour cela, nous avons adopté une stratégie en deux parties. La première partie est de construire une solution réalisable rapidement et la deuxième est d'améliorer la solution obtenue. La première partie ne cherche donc pas une bonne solution, mais seulement une solution réalisable.

2.1.1 Problème du sac-à-dos multiple

Nous voyons ici la résolution du problème de sac-à-dos multiple (*Bin Packing*) par une heuristique. Ce problème se résume à devoir résoudre le problème du sac-à-dos avec plusieurs sacs. Donc il faut remplir des sacs ayant une capacité avec des objets qui ont un poids. Le but est de minimiser le nombre de sacs utilisés. Nous avons fait le choix d'utiliser l'algorithme *First Fit Decreasing* pour résoudre ce problème.

Cet algorithme prend en entrée la liste des clients triés dans l'ordre décroissant de leur demande. Il retourne une liste de listes d'entiers qui représente les indices des clients affectés à chaque véhicule. Il parcourt la liste des clients données en entrée, pour chaque client, il cherche à le placer dans la première tournée où il peut le placer. Si le client ne peut pas être placé dans une tournée, alors on crée une nouvelle tournée afin de le placer dans celle-ci. Ci-dessous le pseudo code de l'algorithme :

Algorithm 1 First Fit Decreasing

Data: clients : la liste des clients dans l'ordre décroissant de leur demande
Result: l : l'affectation des clients à chaque véhicule
initialisation : $l = \emptyset$
for $c \in$ *clients* **do**
  for $t \in l$ **do**
    if *c peut être dans t* **then**
      ajouter c à t
      continuer
    end
  end
  if *c n'a pas été ajouté* **then**
    créer une liste vide t'
    ajouter c à t'
    ajouter t' à l
  end
end

Cet algorithme renvoie la répartition des clients pour chaque véhicule. Il est 2-approché et a une complexité de $O(n\log(n))$ où n est la taille de l'entrée. A la fin de l'exécution de cet algorithme, nous avons la liste des clients de chaque tournée. Nous devons maintenant optimiser l'ordre des clients dans chaque tournée (voir la partie 2.1.2).

2.1.2 Problème du voyageur de commerce

Dans cette sous-partie, nous étudions la résolution du problème du voyageur de commerce (*Travelling Salesman Problem*) par une heuristique. Pour notre problème, il faut résoudre le problème du voyageur de commerce pour chacune des tournées, à savoir trouver un cycle passant par tous les clients de façon à minimiser la somme des distances parcourues. Pour résoudre ce problème, nous avons implémenté l'algorithme *Nearest Insertion*.

Cet algorithme prend en entrée la liste des clients pour chaque tournée (voir partie 2.1.1). Il retourne une liste de listes d'entiers qui représente les indices des clients affectés à chaque tournée dans l'ordre dans lequel la tournée doit être effectuée. Voici le pseudo code de l'algorithme :

Algorithm 2 Nearest Insertion

Data: l : l'affectation des clients à chaque véhicule
Result: l2 : la liste des tournées
for $t \in$ *tournées* **do**
  Choisir un client i le plus proche du dépôt, former la tournée 0-i-0
  Clients de t $\leftarrow$ Client de t $\backslash\{$i$\}$
  while *Clients de* $t \neq \emptyset$ **do**
    k $\in$ Clients de t : $d(k,t) =$min$_{i\in t}$ d(i,t)
    Trouver (i,j) : δ f = c_{ik} + c_{kj} - c_{ij} soit minimal
    Insérer k entre i et j
    Clients de t $\leftarrow$ Clients de t$\backslash\{k\}$
    end
  end

Cet algorithme renvoie la liste des tournées. Il est 2-approché pour chaque véhicule et a une complexité de $O(n^2)$ où n est le nombre de clients par tournée. La solution retournée est loin de la solution optimale. Comme évoqué dans l'introduction de la partie 2.1, nous cherchons une solution réalisable rapidement afin de partir de celle-ci pour en trouver une meilleure. Dans ce but, nous utilisons un algorithme de recuit simulé (voir partie 2.1.4).

2.1.3 Recherche locale

Pour améliorer les tournées, nous appliquons ensuite une recherche locale avec une liste de tabou. Le but de la recherche locale est de réduire le coût de chaque tournée en y permutant deux clients. Nous gardons en mémoire une liste de tabou pour éviter d'être bloqué dans un optimum local.

2.1.4 Amélioration par recuit simulé

A la fin des algorithmes évoqués dans les parties 2.1.1 et 2.1.2, nous obtenons une solution réalisable. Cette solution étant très loin de la solution optimale, il nous faut l'améliorer. Pour cela, nous avons implémenté un algorithme de recuit simulé.

Cet algorithme prend en entrée une solution réalisable et retourne une solution réalisable et proche de la solution optimale. Voici les arguments de cet algorithme :

- k_{max} : Le nombre d'itérations maximal que l'algorithme va faire, elle vaut 2000 dans notre méthode
- T : la température initiale de l'algorithme, elle vaut 1000 dans notre méthode
- α : le coefficient de mise-à-jour de la température, à chaque itération : $T \leftarrow T \times \alpha$
- $energy_{min}$: l'énergie minimale de la solution trouvée, si l'algorithme trouve une solution en dessous de $energy_{min}$, alors il s'arrête

L'algorithme de recuit simulé prend la solution courante, et si le nombre k d'itérations est inférieur à k_{max}, alors il va chercher le voisin de la solution courante. Pour trouver le voisin, on change un client de tournée, on fait une permutation de deux clients dans une même tournée, on ajoute une tournée ou on supprime une tournée vide. Ces façons de trouver un voisin sont tirés aléatoirement avec respectivement 1/3, 1/3, 1/6 et 1/6 de chance d'être pris. Ensuite l'algorithme compare les énergies des solutions. L'énergie est calculée de la manière suivante :

$$\sum_{t \in tournees} distance_t + 100 \times \max\{0, (\#tournee - m)\} \tag{1}$$

Si l'énergie du voisin est meilleure (plus basse), alors on remplace la solution courante par le voisin. Sinon on remplace la solution courante par le voisin avec une probabilité de $\exp(\delta energy/T)$ où $\delta energy$ est la différence entre l'énergie du voisin et l'énergie de la solution courante. De plus, on garde en mémoire la meilleure solution rencontrée, c'est cette solution qui sera retournée. Voici le pseudo code de l'algorithme :

Algorithm 3 Simulated Annealing

Data: s : la solution à améliorer
Result: s' : la meilleure solution rencontrée
initialisation : k = 0 et $energy_s$ = energy(s)
while $k < k_{max}$ *ou* $energy_s < energy_{min}$ **do**
  sn ← voisin(s) et $energy_{sn}$ = energy(sn)
  p est tiré aléatoirement entre 0 et 1
  if $energy_{sn} < energy_s$ *ou* $p < \exp(\delta energy/T)$ **then**
    s ← sn
    $energy_s \leftarrow energy_{sn}$
  end
  if $energy_{sn} < energy_{s'}$ **then**
    s' ← sn
    $energy_{s'} \leftarrow energy_{sn}$
  end
  k ← k + 1
  T ← T × α
end

A la suite de l'exécution de cet algorithme, nous obtenons une solution proche de la solution optimale. Nous avons donc une bonne solution obtenue rapidement à la suite des trois algorithmes décrit dans les parties 2.1.1, 2.1.2 et 2.1.4. Par exemple, nous obtenons, pour l'instance `A-n32-k5`, une solution autour de 1800 à la suite des algorithmes des parties 2.1.1 et 2.1.2, alors que l'on trouve une solution proche de 900 après l'algorithme de recuit simulé. La solution optimale de l'instance `A-n32-k5` est à 787 (voir la partie 2.3).

2.2 Résolution exacte par `PLNE`

Le système d'équations ci-dessous nous permet de modéliser le `VRP` sans la capacité :

$$\text{Min} \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\begin{cases} \sum_{j=1}^{n} x_{0j} \leq m \\ \sum_{i=1}^{n} x_{i0} \leq m \\ \sum_{j=0}^{n} x_{ij} \leq m, \forall i \in \mathcal{N}_{\mathcal{C}} \\ \sum_{i=0}^{n} x_{ij} \leq m, \forall j \in \mathcal{N}_{\mathcal{C}} \\ x_{ij} \in \{0,1\}, \forall (i,j) \in A \end{cases}$$

Nous utilisons ensuite `Cplex` pour résoudre ce système. La solution obtenue est une répartition des clients dans des tournées qui minimise la fonction objectif. Cependant ces tournées ne contiennent pas forcément le dépôt ou dépassent la capacité. Afin de briser ces tournées non

réalisables, nous ajoutons, au fur et à mesure, les contraintes suivantes qui nous permettent de supprimer ces tournées non réalisables.

$$\sum_{i \in W} \sum_{j \in \{0,1,...,n\} \setminus W} x_{ij} \geq \left\lceil \frac{\sum_{i \in W} d_i}{Q} \right\rceil \forall W \subset \{1, ..., n\}, W \neq \emptyset$$

A chaque fois que le solveur détecte une solution valide, nous analysons chacune des tournées de cette solution. Si une tournée ne passe pas par le dépôt, cette contrainte est ajoutée et force la tournée à avoir au moins un arc sortant ce qui a pour conséquence de briser le cycle. De plus, si une tournée passe par le dépôt, nous vérifions que la somme des demandes des clients ne dépasse pas la capacité. Si cette somme dépasse la capacité du véhicule cette tournée n'est pas réalisable. Cette contrainte est alors ajoutée et indique le nombre minimum de véhicules pour servir tous les clients de cette tournée.

2.3 Résultats expérimentaux

Nous analysons ici les premiers résultats de nos deux méthodes de résolution. Pour la résolution approchée, nous affichons ci-dessous les solutions trouvées à différentes étapes pour l'instance `A-n32-k5` :

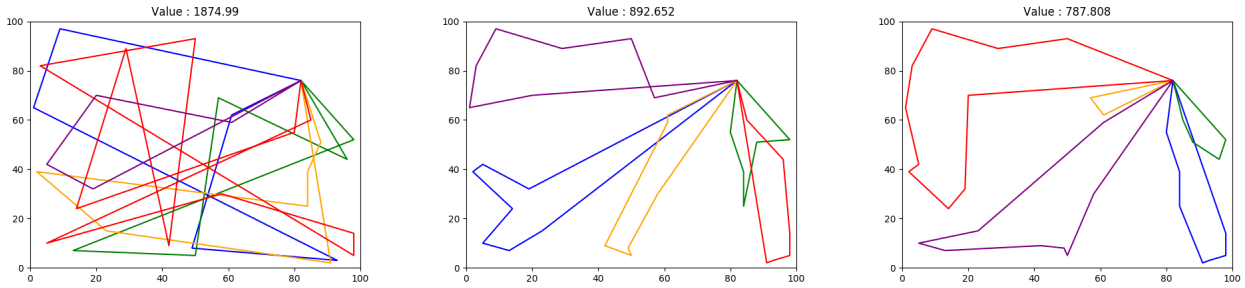
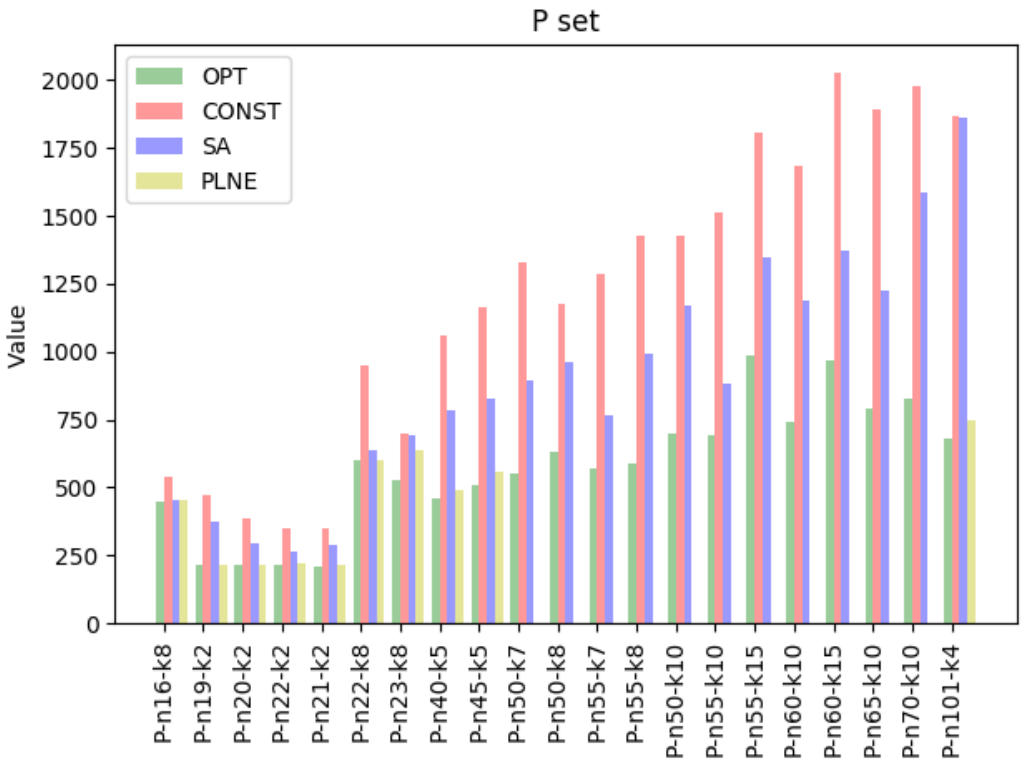


FIGURE 1 – Tournées obtenues par : a) Construction b) Recuit simulé c) PLNE

Dans la figure ci-dessus, nous pouvons voir en a) la solution obtenue après l'étape de construction. Cette solution est très éloignée et a une valeur de 1874. En b), nous avons la solution obtenue après l'algorithme du recuit simulé. Cette solution est bien plus approchée avec une valeur de 892. En c), nous avons la solution optimale avec une valeur de 787. De ces figures, nous pouvons voir que le recuit simulé améliore notre solution de départ et retourne une solution assez proche de la solution optimale. On peut notamment reconnaître des suites de clients dans certaines tournées.

Dans la figure ci-dessous, nous comparons les résultats obtenus par l'heuristique de construction, le recuit simulé et la résolution exacte des instances de type P. Nous limitons le solveur à 300 secondes pour chaque instance.



Du fait de limiter le solveur à 300 secondes, nous remarquons que nous n'obtenons pas de résultat sur les grandes instances. Cela s'explique du fait que le nombre de contraintes de coupes augmente exponentiellement avec la taille de l'instance. Nous constatons que le recuit simulé renvoie des résultats beaucoup plus proche de l'optimum que l'heuristique de construction. Dans les cas plus difficiles, l'heuristique de recuit simulé nous donne des solutions réalisables avec un temps rapide. Pour les grandes instances, la résolution exacte prend plus de temps, mais donne de meilleurs résultats.

3 Deuxième partie

Dans cette partie, nous souhaitons rendre le problème plus proche de la réalité. C'est la raison pour laquelle nous ajoutons des contraintes supplémentaires.

3.1 Ajout du temps de travail

Dans un premier temps, nous avons ajouté des contraintes sur le temps de travail. De ce fait, nous limitons le temps de travail de chaque technicien et nous pénalisons le temps de travail supplémentaire dans la fonction objectif.

De ce fait, nous définissons un temps de travail maximal $d_{max} = \dfrac{1.5 \sum_{j=1}^{n} c_{0j}}{m}$ pour chaque technicien. Pour chaque tournée, nous ajoutons dans la fonction objectif un terme qui est égale à $max(c_{tourne} - d_{max}, 0)$.

Nous remarquons que l'ajout de cet aspect est équivalent au problème initial, sauf que la fonction objectif est modifiée. La résolution du problème avec cette nouvelle contrainte n'est donc pas intéressante. Donc, nous ne l'avons pas étudiée en profondeur.

3.2 Ajout de compétences

Afin d'essayer d'augmenter la complexité du problème et de le rapprocher du réel, nous introduisons ici la compétence des techniciens. Nous disposons de 2 compétences A & B et nous attribuons les compétences aux techniciens de manière suivante :

— $\lfloor 20\% \rfloor$ des techniciens sont des 2 compétences A et B
— $\lfloor 40\% \rfloor$ de compétence A
— $\lfloor 40\% \rfloor$ de compétence B
— le reste n'a pas de compétence spécifique

Les clients ont aussi des demandes spécifiques aux différents services. Voici la répartition des clients :

— 15% des deux services A & B
— 30% du service A
— 30% du service B
— 15% pas de service

Des relations existent entre les différentes compétences. En effet, un client ayant besoin les deux services ne peut être satisfait que par un techniciens ayant les deux services. De plus, un client ayant besoin du service A (respectivement B) doit être servi par un technicien ayant le service A (respectivement B) ou les deux services A & B. Les clients n'ayant pas besoin de compétence peuvent être servi par n'importe quel technicien.

Nous pénalisons dans la fonction objectif le fait qu'un client soit servi par un mauvais technicien par un terme qui est 3 fois la distance entre le dépôt et ce client. Par exemple, un client A servi par un technicien B ou un client A & B servi par un technicien B.

Nous avons modifié la structure de données pour qu'elle prenne en compte ces compétences. Pour cela, comme dans la partie 2.1.1, nous trions les clients en ordre décroissant de la demande. En suite, nous mettons les clients de type A & B dans les tournées A & B, si le nombre de tournées de ce type n'est pas suffisant, nous mettons le reste des clients A & B de côté. La même procédure est appliquée pour les clients de type A (respectivement B) avec les tournées A (respectivement B). Enfin, tous les clients restant sont placés dans les tournées sans prendre en compte les compétences avec l'algorithme *First Fit Decreasing*.

Les étapes suivantes de l'heuristique sont inchangées car elles évaluent les tournées avec la fonction objectif qui est déjà implémentée et qui prend en compte les pénalités des clients mal servi.

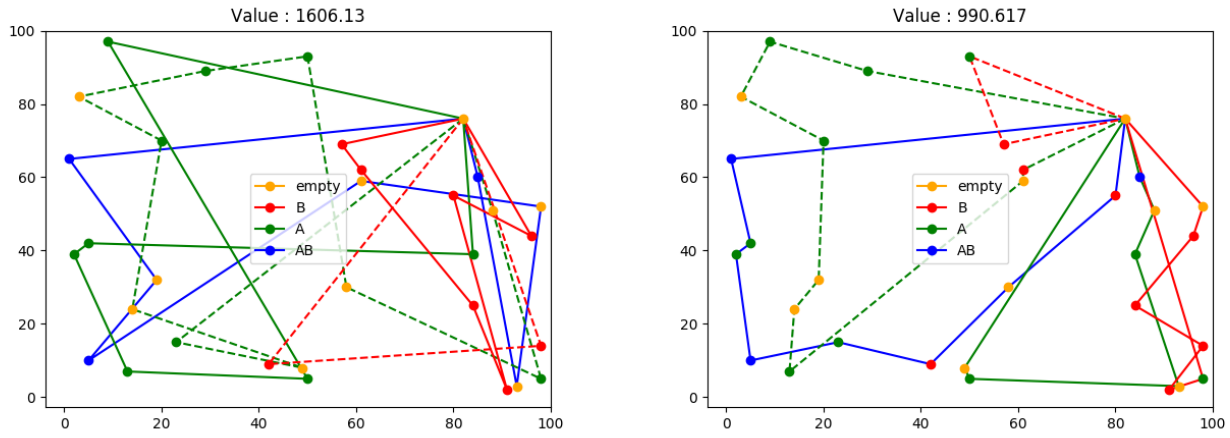


FIGURE 2 – Tournées obtenues par heuristiques : a) Construction b) Recuit simulé pour `A-n32-k5`

Nous observons dans la figure ci-dessus les solutions retournées à différentes étapes de l'heuristique. En a), nous pouvons voir la solution retournée par l'heuristique de construction, elle a une valeur de 1606. En b), nous avons la solution retournée par le recuit simulé avec une valeur de 990. Ce que nous pouvons remarquer dans cette figure c'est que la valeur de la solution est meilleur après l'application du recuit simulé. Nous constatons que les clients sont globalement attribués dans les service correspondant.

Ensuite, nous avons modifié la PLNE pour qu'elle soit adaptée à ce problème. En ajoutant une troisième dimension k aux variables x_{ij}. Nous fixons le nombre de tournées à m. Une autre variable, p_{0j}^k, est ajoutée dans la fonction objectif, elle est décrite de la manière suivante :

$$p_{0j}^k = \begin{cases} \text{distance entre le dépôt et le client j s'il est servi dans la mauvaise tournée k} \\ 0 \text{ sinon} \end{cases}$$

La PLNE devient :

$$\text{Min} \sum_{(i,j)\in A} (c_{ij}^k + 3p_{0j}^k)x_{ij}^k$$

$$\begin{cases} \sum_{j=1}^{n} x_{0j}^k = 1, \forall k \in \{0, ..., m\} \\ \sum_{i=1}^{n} x_{i0}^k = 1, \forall k \in \{0, ..., m\} \\ \sum_{k=0}^{m} x_{0j}^k \leq 1, \forall j \in \{0, ..., n\} \\ \sum_{k=0}^{m} x_{i0}^k \leq 1, \forall i \in \{0, ..., n\} \\ \sum_{j=0}^{n} x_{ij}^k \leq 1, \forall i \in \{0, ..., n\}, \forall k \in \{0, ..., m\} \\ \sum_{i=0}^{n} x_{ij}^k \leq 1, \forall j \in \{0, ..., n\}, \forall k \in \{0, ..., m\} \\ \sum_{k=0}^{m} \sum_{j=0}^{n} x_{ij}^k = 1, \forall i \in \{0, ..., n\} \\ \sum_{k=0}^{m} \sum_{i=0}^{n} x_{ij}^k = 1, \forall j \in \{0, ..., n\} \\ x_{ij}^k \in \{0, 1\}, \forall (i, j) \in A, \forall k \in \{0, ..., m\} \end{cases}$$

Ces contraintes modélisent l'ensemble du problème mais elles ne sont pas suffisantes pour avoir plus d'information sur chaque tournée. Les valeurs retournées dans la fonction *callback* ne nous permettent pas de couper les points.

Avec cet aspect, le problème devient plus facile pour les heuristiques. En effet, le problème est moins "combinatoire" parce que le seul moyen de résoudre ce problème est de placer au mieux les clients dans les tournées correspondantes à leur besoin.

4 Conclusion

Ce projet nous a permis de mettre en pratique les connaissances acquises en cours, comme les algorithmes de coupes. Le projet est réaliste et peut très bien s'appliquer dans la réalité.

De plus, il met en évidence les différences entre les résolutions approchées et les résolutions exactes. En effet, la résolution approchée est rapide mais ne renvoie pas la solution optimale. En revanche, la résolution exacte est plus longue, mais retourne la solution exacte. Nous pensons donc que les deux méthodes doivent être utilisées selon les conditions. Si nous disposons d'assez de temps, la résolution exacte reste la meilleure option. Au contraire, si le temps nous manque, la résolution approchée est plus adaptée.