Rapport Méta Heuristiques : Une solution du problème de planification d'évacuations en cas d'incendie

Le Minh TRAN, Trong Hieu TRAN [1]
Tutoré par : Mme Marie-Jo HUGET [2]
[1] Élève ingénieur en 4IR, Département GEI, INSA Toulouse, FRANCE
[2] *Enseignant-Chercheur au LAAS-INSA, Toulouse, FRANCE*

26 Mai 2019

Abstract

Dans ce rapport, nous résumons plusieurs tests and résultats obtenus du développement d'un programme en utilisant des méthodes de Méta Heuristiques. L'objectif de ce programme est de résoudre le problème de la planification d'évacuations pour sauver des vies humaines en cas d'incendie avec la méthode de Recherche Locale. Ces travaux est un projet dans la matière Méta Heuristiques (UF Systèmes Intelligences) à l'INSA de Toulouse.

Contents

1 Introduction

Ce TP a pour but de résoudre un problème d'optimisation combinatoire en utilisant des heuristiques et méta heuristiques.

Le problème à résoudre porte sur la planification d'évacuations pour le sauvetage en cas d'incendie. Ce sujet s'inspire d'études au LASS (equipe ROC) dans le cadre d'un project collaboratif (GeoSafe) se déroulant entre la France et l'Australie, plus d'information se trouve ***ici***.

Ce rapport contient le travail sur le sujet de TP Méta Heurisitiques de TRAN Trong Hieu et TRAN Le Minh, programmé en langage Python. Les principes et les algorithmes seront présentés en langage naturel et éventuellement en pseudo-code dans ce rapport; pour voir les codes source en Python, veuillez consulter le dépôt GitHub sur ce projet : `https://github.com/kuro10/Projet_Meta_Heuristiques`.

2 Contexte du problème

2.1 Compréhension du problème

Ce problème aborde la planification d'évacuation pour sauver des vies humaines en cas d'incendies. On suppose que la zone à évacuer et que le modèle de propagation d'incendie sont connus. Notons qu'une fois que l'évacuation a débuté pour un secteur d'évacuation, on suppose qu'elle ne puisse pas être interrompue et que son taux d'évacuation reste constant. Alors l'objectif de ce problème reste à trouver une plan d'évacuation dans un délai minimal en respectant des contraints concernant le taux d'évacuation maximal, la capacité des arcs, la date limite de chaque arc, etc. Plus des informations se trouvent dans le sujet de TP[1].

2.2 Représentation des instances

Des instances sont fournies par un générateur qui est décrit plus précisément en [2]. En générale, ils ont de la forme suivant :

- <nombre de sommets à évacuer> , <identifiant de somment sécurisé>
- pour chaque sommet à évacuer :<son identifiant>, <nombre de personne à évacuer>,<taux d'évacuation maximum>,$< k >$,$< v_1, ..., v_k >$, avec k est la longueur du chemin d'évacuation.
- <nombre de sommets>,<nombre de arcs>
- pour chaque arc dans ce graphe : <noeud 1> <noeud 2> <date limite d'utilisation> <durée de traversé> <capacité>

2.3 La structure du projet

Notre répertoire de travail possède les éléments suivants:

- Instances : contenant de différentes scénarios d'évacuations avec des noeuds à évacuer, le plan global et le plan d'évacuation sous forme des graphes
- Solutions : contenant des solutions pour les scénarios décrites dans le dossier *Instances*
- DataProcess.py : contenant des algorithmes d'extractions de données à partir des fichiers instances et solutions, d'analyse et de traitement de données. Il contient aussi des méthodes pour chercher les bornes inférieures et supérieures
- SolutionChecker.py : contenant le vérificateur de solution
- LocalSearch.py : contenant les algorithmes de Recherche Locale et sa version intensive.
- wildfire_main.ipynb : le Python Notebook à exécuter les algorithmes et générer les solutions.

3 Déroulement du projet

3.1 Préliminaire

Cette section destine aux remarques/analyses/traitement préliminaires que nous avons effectués pendant le déroulement du TP et voulions aborder avant les parties principales du rapport.

- Pour l'extraction du graphe depuis des fichiers instances, on crée un nouveau graphe qui ne contient que des arcs figurés dans le plan d'évacuation. Donc, pour un scénario où il y a n noeuds d'évacuations et le chemin d'évacuation de chaque noeud a au plus m arcs, la complexité pour un parcours entier de ce nouveau graphe sera $o(n * m)$
- De ce fait, les algorithmes qui exigent un parcours de graphe auront comme complexité $o(n * m)$, par exemple algorithmes pour extraire les informations d'un arc.
- D'ailleurs, on remarque qu'un parcours dans la liste de noeuds d'évacuations a pour complexité $o(n)$ et un parcours dans un chemin d'évacuation a pour complexité $o(m)$
- Pour calculer la durée totale d'évacuation d'un noeud, il faut parcourir tous les arcs dans son chemin d'évacuation, et à chaque parcours il faut aussi parcourir le graphe pour prendre les longueurs des arcs d'évacuations. Donc la complexité sera $o(nm^2)$

3.2 Vérificateur

Le vérificateur consiste à vérifier une solution selon les critères suivantes :

- Respect des contraintes : une solution doit respecter le taux maximal de l'évacuation et sera invalide si le taux d'évacuation dépasse le taux maximal
- Réalisabilité de la solution : une solution ne doit pas présenter des situations où le nombre total de personnes en train d'être évacuées sur un arc dépasse la capacité maximale de cette arc, sinon cette solution devient invalide
- Exactitude de la valeur de la fonction objective : la fonction de cette valeur objective doit correspondre à la valeur maximale de la durée totale d'évacuation de chacun noeud prenant en compte la date de début, la longueur du chemin, le nombre d'unité de temps pour évacuer toutes les personnes.

Une solution valide doit respecter tous les critères ci-dessus. Dans ce TP Méta Heuristiques , une solution a de la forme suivant [1]:

- <Nom de l'instance résolue>
- <Nombre de sommet à évacuer >
- pour chaque sommet à évacuer : <son identifiant>, <son son taux d'évacuation>,<sa date de début d'évacuation>
- ....

Afin de savoir si une solution est valide ou pas, on a proposé une fonction qui peut vérifier tous les contraints ci-dessus . En fait, dans cette algorithme du vérificateur, on fait un parcours de graphe pour créer tout d'abord un tableau de ressource pour représenter la capacité disponible actuelle des arcs d'évacuations, avec deux dimensions: une pour le temps d'écoulement et autre pour les arcs d'évacuations. Puis on vérifie pour chaque noeud d'évacuation s'il respecte la contrainte de taux maximal par comparer avec les informations fournies dans le fichier source. Ensuite on vérifie avec des autres noeuds d'évacuations s'il y a des conflits dans la capacité des arcs pendant un même moment en

remplissant le tableau de ressource avec la capacité restante après avoir pris le nombre de personnes actuellement dans un arc, et une valeur de capacité restante négative signifie l'erreur de la solution. Après, on peut aussi vérifier les contraints sur la date limite de chaque arc passé et re-évaluer la fonction objective de cette solution.

Il nécessite donc un parcours du chemin d'évacuations pour obtenir les noeuds passés et dans chacun des ces parcours un autre parcours de graphe pour obtenir les informations d'un arc. Ces parcours imbriqués rendent une complexité de $o(n^2m^2)$, plus la complexité de la création du tableau de ressource de $o(nm)$, l'algorithme donne en tout une complexité de $o(n^2m^2)$.

Algorithm 1 Solution Checker Algorithm

```
Data : an instance file, a solution file
Result : a boolean which reponds the question "Is this solution is valid or not?"
begin
Read data file
Read solution file                                                    ▷ Initialization
Generate a set of ressources R which correspond to each edges in the graph
for each evacuation node do
    if evacuation rate > max rate then
        Print('ERROR on evacuation rate !')          ▷ Check evacuation rate of this node
        return False
    for each edges of evacuation path do
        Update ressource of this edge
        if ressource of this edge is overload then
            Print('ERROR on capacity of edge !')    ▷ Check capacity constraint on this edge
            return False
        if start + duration > due date then
            Print('ERROR on due date of edge !')    ▷ Check due date constraint on this edge
            return False
    Compute evacuation time of this node
Compute total evacuation time by finding max end time
if total evacuation time != Objectif function then
    Print('ERROR on objectif function !')                  ▷ Verify objectif function
    return False
return True
```

Le vérificateur de solution est planifié pour être la première fonctionnalité à programmer. En effet, il est très important puisqu'il va nous aider à vérifier si notre solution obtenu depuis notre méthode de résolution est valide ou pas, et puis on peut apporter des modifications et des améliorations nécessaires à temps.

3.3 Bornes inférieures et supérieures

Comme il s'agit d'un problème de minimisation, il est important de trouver un intervalle dans lequel se trouve la solution optimale. D'où l'importance de la borne inférieure - le seuil où la solution ne passe jamais au dessous, et la borne supérieure - une solution non optimale trouvée à partir d'une certaine heuristiques.

3.3.1 Borne inférieure

Le calcul de borne inférieure consiste à trouver la durée la plus grande parmi la durée pour évacuer pour chaque noeud d'évacuation seul. Ainsi, on va calculer la durée nécessaire à évacuer chaque noeud

en faisant la somme de la longueur totale d'un chemin d'évacuation et le nombre d'unité de temps pour évacuer toutes les personnes de ce noeud, puis choisir la valeur maximale entre eux.

Dans cet algorithme, on calcule la durée d'évacuation de chaque noeud d'évacuation, d'où un parcours de liste de noeuds d'évacuation, et le calcul de cette durée nécessite un parcours de chemin d'évacuation de chaque noeud, dans lequel il y aura un autre parcours de graphe pour extraire les données d'un arc d'évacuation. La complexité en tout est donc $o(n^2m^2)$.

Algorithm 2 Compute Lower Bound Algorithm

Data : *An evacuation tree T, a graph G*
Result : *An integer which indicates an upper bound of problem*
begin
$TabEndtime \leftarrow []$
for *each node in list of evacuation nodes* **do**
  $evaRate \leftarrow$ maxRate
  $travelTime \leftarrow 0$
  for *each edge on the evacuation path* **do**
    if *edge capacity* $<$ *evaRate* **then**
      $evaRate \leftarrow$ edge capacity ▷ evaRate must respect edge capacity constraint
    $travelTime \leftarrow travelTime +$ length of edge
  $travelTime \leftarrow travelTime +$ ceil(nbEvacuees / $evaRate$) ▷ Finish time of the last person
  Save $travelTime$ in $TabEndtime$
$lowerBound \leftarrow$ element max of $TabEndtime$
return *lowerBound*

3.3.2 Borne supérieure

Nous avons cherché la borne supérieure en utilisant une fonction de calcul du temps de fin d'évacuation, ce qui est implémentée à l'aide d'une heuristique glouton qui consiste, à partir d'un arrangement des noeuds d'évacuation, à trouver une date de début convenable à l'ordre d'évacuation donnée. Donc, le résultat obtenu va dépendre de l'ordre d'évacuation des noeuds. En réalité, le caractère d'optimalité non garantie du résultat trouvé est partiellement reflété sur le fait que l'on ne peut pas être sûr pour quelle ordre d'évacuation on obtiendra le meilleur résultat. Pour notre approche actuelle, nous avons arrangé la liste d'évacuation selon ordre de durée d'évacuation totale décroissante, car à notre avis, pendant le ravage progressif de l'incendie, les chemins d'évacuations seraient de plus en plus dégradé et il serait très difficile d'évacuer des noeuds dont la durée totale d'évacuation est longue, donc on donne la priorité aux noeuds avec longue durée d'évacuation. Il ne reste qu'à appliquer la fonction de calcul de temps de fin d'évacuation sur l'ordre d'évacuation ci-dessus.

Algorithm 3 Compute Upper Bound Algorithm

Data : *An evacuation tree T, a graph G*
Result : *A solution S with endtime corresponding*
begin
Find a *priorityList* of evacuation nodes
$endtime, solution \leftarrow$ **Compute End Time** $(priorityList, T, G)$
return *endtime, solution*

Effectivement, pour la fonction de calcul de temps de fin, nous avons repris la même implémentation que celle du vérificateur de solution, avec un petit changement : pendant l'analyse de l'évacuation d'un noeud, quand on détecte une valeur négative au niveau de capacité restante sur un arc, nous allons décaler le début de l'évacuation du noeud courant en fonction de cette valeur négative, donc ce noeud

en cours d'analyse va éviter le conflit avec l'évacuation des autres noeuds. Puis il faut aussi refaire une vérification pour voir si ce décalage du temps crée de nouveaux conflits pour les autres noeuds, et effectuer à nouveau un autre décalage si nécessaire. Au pire cas on pourrait parcourir toute la taille t de la dimension du temps d'écoulement du tableau de ressource pour trouver un décalage convenable.

Algorithm 4 Compute End Time Algorithm

Data : *an ordered list of evacuation nodes L, an evacuation tree T, a graph G*
Result : *a full solution S, end time*
begin
Generate a set of ressources R which correspond to each edges in the graph
for *each evacuation node in L* **do**
    Apply maximum evacuation rate
                                                ▷ Find the best delay time
    $delay \leftarrow 0$
    while *capacity constraint is not ok* **do**
                                                ▷ Shift the task until there is no overload
        for *each edge on the evacuation path* **do**
            Try to use the ressource on this edge
            $invalidRessource \leftarrow$ total duration where there are errors on capacity constraint
            $delay \leftarrow delay + invalidRessource$
                                                ▷ Apply delay time as start time of this node
                                                ▷ There will be no capacity constraint with this start time
    $start \leftarrow delay$
    for *each edge on the evacuation path* **do**
        Use and update the ressource on this edge
        Save the information of this node on this edge (start,end,duration,rate,due date,..)
$endtime \leftarrow$ the last endtime, for all evacuation node
Generate a full solution S which contains triplets $[nodeId, startTime, evacuationRate]$
return *endtime*, S

En bref, l'algorithme de calcul de temps de fin d'évacuation a une complexité de $o(tn^2m^2)$. L'algorithme d'arrangement d'ordre d'évacuation doit parcourir la liste des noeuds d'évacuation, puis faire le calcul de durée d'évacuation pour chaque parcours, donc il a une complexité de $o(n^2m^2)$. L'algorithme de calcul de borne supérieure a donc une complexité de $o(tn^2m^2)$

3.4 Recherche Locale

3.4.1 Intensification

Jusqu'à maintenant, on possède une fonction qui peut fournir des solutions dont l'optimalité n'est pas garantie, c'est-à-dire les bornes supérieures de la fonction objective. La prochaine démarche consiste à améliorer la solution grâce à la méthode de recherche locale incluant un processus d'intensification avec méthode de descente.

La première étape c'est de définir des voisinages pour explorer l'espace de recherche. Comme notre problème actuel c'est de trouver une ordre d'évacuation A qui pourrait donner un résultat optimal, un voisinage de A sera cette même ordre mais avec deux noeuds quelconques ayant leurs priorités permutées.

On peut maintenant procéder à l'implémentation de l'algorithme de recherche locale: à partir d'une liste de noeuds d'évacuation, on cherchera le voisinage de cette liste pour trouver le voisin qui donne le meilleur résultat, et puis on répète cette même étape pour les itérations suivantes en cherchant le

voisinage de ce voisin, trouvant le voisin qui donne le meilleur résultat et ainsi de suite, jusqu'à un nombre maximal prédéfini d'itérations. Tout d'abord, on choisit la solution de la borne supérieure comme la liste de noeuds d'évacuation initial. De plus, on est sûr qu'il s'agit d'une méthode de descente car on commence par la solution d'une borne supérieure et, après chaque itération, on ne retient que l'ordre d'évacuation donnant le résultat le plus optimal, alors pendant toutes les itérations, on ne peut que voir le temps de fin d'évacuation avoir tendance à se diminuer et diminuer depuis une borne supérieure. Il est attendu qu'on obtienne un minimum au moins local à l'issue de cet algorithme.

Pourtant, de cette façon, pour chaque liste d'évacuation de n noeuds, il y aura C_n^2 voisins possibles. Comme il faudra évaluer à chaque itération tous les voisins d'une liste pour trouver l'ordre la plus convenable qui donne le résultat le plus optimal, il y aura au total beaucoup de cas à évaluer. Il faut donc trouver une façon pour réduire la taille de voisinage.

Algorithm 5 Generate List of Neighbors Algorithm

Data : *A list of evacuation nodes E*
Result : *A list of neighbors of the input list*
begin
neighborList ← []
for *each node i in E* **do**
  for *each node j after i in E* **do**
    if *i and j are conflicts* **then**
      neighbor ← *E*
      Swap *i* and *j* in *neighbor*
      Save *neighbor* in *neighborList*
return *neighborList*

En effet, nous avons découvert qu'il existe des voisins d'une ordre qui ont évidemment le même temps de fin d'évacuation que celui de cette ordre. Plus précisément, la permutation de deux noeuds dont les évacuations ne rencontrent pas des conflits sur la capacité d'arc ne change pas le temps de fin obtenu pour cette ordre d'évacuations. Ainsi, nous avons pensé à une façon pour réduire la taille de voisinage en ne faisant que des permutations pour deux noeuds qui ont potentiellement des conflits. "Conflits entre deux noeuds" ici signifie que les chemins d'évacuations de ces deux noeuds partages quelques arcs communs et que pour tout moment, il est possible que la capacité de ces arcs soit dépassé (on va vérifier donc si la somme des taux d'évacuations de ces deux noeuds est supérieure à la capacité de ces arcs).

En outre, parfois il semble qu'on tombe déjà sur un minimal local après une certaine itération où, même si on continue à effectuer les itérations restantes, le résultat obtenu ne changerait pas. De ce fait, on décide d'ajouter une condition d'arrêt où, à côté du nombre maximal d'itérations, si on observe un certain nombre de fois consécutives où le meilleur résultat courant ne change pas, on en déduit qu'on a déjà le minimal local et on arrête l'algorithme.

Algorithm 6 Local Search Algorithm

Data : *An initial solution S_0, an evacuation tree T, a graph G, number of iteration maximum N*
Result : *Best end time, best solution*
begin
$orderedList \leftarrow$ create an ordered list from initial solution S_0
$bestEndtime, bestSolution \leftarrow$ **Compute End Time** $(orderedList, T, G)$ ▷ Initialization
$ite \leftarrow 0$
while $ite < N$ **do**
  $neighborList \leftarrow$ **Generate List of Neighbors** $(orderedList)$ ▷ Find all neighbors
  for *each neighbor x in neighborList* **do**
    $end, currentsolution \leftarrow$ **Compute End Time** (x, T, G)
    if $end < bestEndtime$ **then**
      $orderedList \leftarrow x$ ▷ Find the best neighbor
      $bestEndTime \leftarrow end$
      $bestSolution \leftarrow$ current solution
  $ite \leftarrow ite + 1$
return $bestEndtime, bestSolution$

Pour récapituler, voici les fonctionnalités développées pour cette partie:

- Un algorithme qui fournit en sortie une liste des noeuds non conflit avec un noeud donné en entrée. Cet algorithme parcourt la liste de noeuds d'évacuation et vérifie si le chemin d'évacuation du noeud en entrée a des arcs en commun avec celui du noeud évalué, si c'est le cas on vérifie la somme des taux d'évacuations de ces deux noeuds est supérieure à la capacité de ces arcs, qui nécessite donc un parcours des listes des arcs communs (donc complexité $o(m)$) dans lequel on fait aussi un parcours de graphe pour obtenir la capacité maximale de chaque arc à évaluer. En tout, cet algorithme a une complexité de $o(n^2m^2)$.

- Un algorithme de recherche du voisinage d'une liste qui ne prend pas en compte des éléments non conflits de cette liste. On fait ici un parcours de liste des noeuds d'évacuations, puis pour chaque noeud on cherche la liste des noeuds non conflits avec le noeud actuellement évalué, ensuite on ne génère que des voisins de cette liste par permutation du noeud actuel avec un autre noeud de la liste si et seulement si cet autre noeud ne se trouve pas dans la liste des noeuds non conflits avec le noeud actuel. La complexité de cet algorithme sera $o(n^3m^2)$

- Et finalement, l'algorithme principal de la recherche locale utilisant tous ces deux fonctionnalités ci-dessus. Pendant I itérations au maximum, on va parcourir un voisinage de taille maximale de C_n^2 donc de l'ordre n^2, et à chaque parcours on applique l'algorithme de calcul de temps de fin d'évacuation. En tout, l'algorithme aura une complexité de $o(Itn^4m^2)$.

Malgré une complexité qui semble très haute, elle ne reflète que des pires situations où on doit tenir compte de tous les cas possibles. En réalité, on a réduit la taille de voisinage et limité autant que possible le nombre d'itération en ajoutant une nouvelle condition d'arrêt, par conséquent pendant la plupart du temps on devrait avoir un temps de calcul beaucoup plus rapide.

3.4.2 Diversification

Il s'agit d'une finalisation de l'algorithme précédente qui utilise un processus de diversification dans l'espoir de trouver l'optimum global.

Algorithm 7 Local Search with Random Start Algorithm

Data : *Number of iteration maximum* N_{it}*, number of start point* N_{sp}
Result : *Best end time, best solution*
begin
$bestEndtime \leftarrow 9999$ ▷ Initialization
$bestSolution \leftarrow []$
for *i in range*(N_{sp}) **do**
  Randomize an ordered list of evacuation nodes L_r ▷ Random a start point
  $\_\_$, $InitSolution \leftarrow$ **Compute End Time** (L_r)
  $endtime, solution \leftarrow$ **Local Search** $(InitSolution, N_{it})$ ▷ Find an optimum solution
  if $endtime < bestEndtime$ **then**
    $bestEndtime \leftarrow endtime$ ▷ Choose the best optimum solution
    $bestSolution \leftarrow solution$
return $bestEndtime, bestSolution$

Pour cette partie, nous avons décidé de choisir le processus multi-start. Ainsi, on aura dans cet algorithme le nombre de points de départ P, et chacun de ces points de départ sera l'ordre d'évacuation aléatoirement arrangée avec laquelle on appliquera l'algorithme de recherche locale, au lieu d'une solution de la méthode pour chercher la borne supérieure, générée selon une ordre de priorité de durée d'évacuation décroissante. Autrement dit, on appliquera la méthode de descente sur P des solutions choisies aléatoirement dans l'ensemble des solutions réalisables. On a au final un algorithme de complexité $o(PItn^4m^2)$, avec P le nombre de points de départ, I le nombre d'itérations de recherche local, t la taille de la dimension du temps d'écoulement du tableau de ressource, n le nombre de noeuds d'évacuations et m le nombre maximal d'arcs d'un chemin d'évacuation.

4 Tests and Résultats

Attention: Les tableaux de résultats sont disponibles en Annexe

4.1 Bornes inférieures et supérieures

4.1.1 Borne inférieure

A côté du résultat des bornes inférieures figuré dans 1, on observe que, avec notre approche actuelle de décaler le début d'évacuations des noeuds, on pourrait obtenir le résultat égale à la borne inférieures seulement si tous les noeuds démarraient à zero. De ce fait, on a créé des fichier solutions pour cela, avec le début de tous les noeuds à zero et la fonction objective prend la valeur de la borne inférieure. Effectivement, ces solutions sont d'autant plus invalides que notre approche actuelle est juste de décaler le début et non de modifier les taux d'évacuations. Les résultats de validité détaillés sont disponible dans la section 3.3.

4.1.2 Borne supérieure

En observant le résultat obtenu sur les bornes supérieures figuré dans 2, tout d'abord on trouve que les valeurs sont bien supérieures aux bornes inférieures. De plus, leurs solutions passent aussi le test de validité (précisé dans la section 3.3), donc réalisables. On a pu confirmer à ce moment-là que nous avons une fonction de calcul du temps total de fin d'exécution qui fonctionne correctement.

4.2 Recherche Locale

4.2.1 Intensification

A travers 3, on peut observer que le temps d'exécution est relativement long dû au fait que nous avons choisi au plus 10 itérations pour le calcul. Concernant les résultats, on a obtenu les valeurs de temps total d'évacuation qui sont bien supérieures aux bornes inférieures, réalisables (vérifiées avec vérificateur de solution - à consulter la section 3.3), et d'ailleurs assez proches aux bornes supérieures qu'on a trouvé basé sur l'arrangement de la liste des noeuds d'évacuation selon l'ordre de durée d'évacuations décroissante, avec une différence moyenne entre la borne supérieure et le résultat de 9. En complément avec la section 3.1.2, on trouve ensuite que notre approche pour la priorité d'évacuation pourrait être assez raisonnable en vraie vie au cas où une de solution optimal pour l'évacuation n'était pas encore disponible. Néanmoins, ces résultats pourraient être bien juste des optimaux locaux, c'est pourquoi on va procéder au processus de diversification.

4.2.2 Diversification

Dans cette partie, on applique la méthode de recherche locale sur 5 points choisis aléatoirement dans l'ensemble de solutions, autrement dit sur 5 listes d'évacuation arrangées aléatoirement. Ceci explique un temps d'exécution considérable pour chaque scénario. Au niveau de résultat, on aperçoit dans 4 des changements de résultat pour deux scénarios, donc le processus de diversification nous a aidés à trouver des solutions plus optimales ! Ainsi, à l'issue de ce processus, on peut être plus sûr sur l'optimalité de la solution obtenue.

Pourtant, est-ce que le résultat trouvé est déjà optimal ? En réalité, la méthode qu'on a utilisée est la méthode approchée où on essaie de trouver un résultat qui tend le plus vers la borne minimale avec plusieurs techniques. Dans notre cas, il peut exister beaucoup de zones de résultats où il y aurait une solution potentiellement optimale et qui ne sont pas encore exploitées. Pour cela, nous avons deux futures approches: soit on augmente les points de départ de la méthode multi-start tout en réduisant la complexité de l'algorithme, de peur d'un extra long temps d'exécution, avec les techniques discutées plus tard dans ce rapport, soit on implémente la méthode de changement du taux d'évacuation en espérant d'obtenir de meilleurs résultats.

Pour ce moment, avec chaque scénario 10 noeuds d'évacuation, on peut parvenir à 10! arrangements de l'ordre d'évacuations, et concernant ce que nous avons effectué pour ce TP, nous avons considéré la méthode de recherche locale sur 5 solutions initiales différentes, chaque point concerne au plus 10 itérations d'application de la méthode de descente à partir d'une solution initiale, chaque fois on peut explorer au plus $C_1 0^2$ solutions voisines. On n'a atteint même pas à 0.1% de nombre total de solutions possibles !

4.3 Vérificateur

Après avoir conçu le vérificateur et généré les solutions de chaque étape, nous l'avons utilisé pour vérifier la validité de nos solutions dans 4 catégories : solution avec temps de début tout à zero, solution pour l'arrangement selon ordre de durée d'évacuation décroissante, solution de la méthode de recherche locale intensifiée, solution de la méthode de recherche locale diversifiée. Nous avons aussi utilisé en complément le vérificateur en ligne fourni par nos encadrants de TP pour garantir le bon fonctionnement de notre version.

Notre vérificateur peut aussi supporter la détection du non-respect de la date limite des arcs, qui est actuellement désactivée (mise en commentaire). La validité des solutions, illustrée dans 5 ne tient donc pas compte de la contrainte de date limite. Pour tous les solutions qui ont utilisé la fonction de calcul de temps de fin d'évacuation, à savoir tout sauf celle avec le temps de début à zéro, elles sont

tous validées. On reconfirme aussi le fait qu'il est impossible d'obtenir un résultat égal à la valeur de borne inférieure avec l'approche de décalage du temps de début pour chaque noeud d'évacuation.

5 Discussion

5.1 Intégration des contraintes sur la date limite des arcs d'évacuation

Pendant la modélisation du sujet d'évacuation en cas d'incendie, à côté de la contrainte de capacité maximale de chaque arc, il existe aussi une contrainte sur la date limite des arcs. En vraie vie, cette date limite représente le ravage progressif d'un incendie où après une certaine date limite, un arc commence à être détruit par le feu et il devient impassable après l'instant date limite + longueur de l'arc.

Nous avons réfléchi à plusieurs approches pour cette nouvelle contrainte. Notre première approche, convenable bien à la méthode de calcul de temps de fin d'exécution que nous avons implémentée précédemment, consiste à trouver le temps de début d'évacuation le plus tard possible à partir duquel le dernier groupe de personnes évacué depuis un noeud peut passer par tous les arcs du chemin d'évacuations tout en satisfaisant la contrainte de temps limite. Donc comment faire pour calculer le temps de début maximal pour chaque noeud ? Une solution c'est de calculer récursivement ce résultat à partir de l'arc entre le dernier noeud du chemin et le noeud sécurisé, prendra comme meilleur résultat courant le minimum entre la date limite de l'arc courant moins la longueur de l'arc précédent et la date limite de l'arc précédent. Le résultat final sera le résultat obtenu après tous les itérations moins l'unité de temps pour évacuer toutes les personnes d'un noeud.

Pourtant, comme ce qu'on a fait dans la fonction de calcul de temps de fin d'exécution est de décaler le temps de début pour chaque noeud, il peut se passer que ce décalage viole la contrainte de temps de début le plus tard que nous avons trouvé avec l'approche ci-dessus. Donc on procède à la deuxième approche: à réduire le taux d'évacuation de chaque noeud.

Comme résultats expérimentaux, on a essayé au niveau algorithmique pour la première approche, même si cela ne donne pas vraiment des solutions pertinentes, alors que la deuxième approche nécessite encore des idées dessus.

5.2 Comment réduire la complexité des algorithmes ?

A travers l'implémentation des algorithmes ci-dessus, on peut observer que la complexité est trop lourde. En effet, c'est le parcours du graphe pour extraire les informations sur des arcs qui est répété la plusieurs fois. En conséquence, il vaut mieux stocker ce graphe dans un tableau de deux dimensions qui rendra l'accès aux informations des arcs plus direct : il ne faut qu'indiquer les identités des noeuds pour extraire les informations sur un arc. Par exemple pour avoir les informations sur l'arc 10-20 il suffit de faire *graphe*[10][20]. Néanmoins, comme l'identité d'un noeud n'est pas compatible à l'indice d'un tableau, il faut chercher une façon pour assimiler les identités des noeuds à une suite des nombres entiers à partir de 1. Une solution envisagée c'est d'utiliser le type de donnée "dictionnaire" pour donner des alias aux identités. Cette méthode dite vectorisation des données du graphes nécessite la conversion des identités des noeuds aux indices de tableau, puis il ne reste qu'à remplir ce tableau avec des informations des arcs du graphe. Si une telle implémentation était effectuée, on pourrait témoigner une optimisation remarquable au niveau de complexité algorithmique.

Et puis, dans la complexité de l'algorithme final on voit apparaître aussi un t dedans qui est la taille de la dimension du temps d'écoulement du tableau de ressource. En effet, on ne doit forcément pas atteindre la taille maximale de cette dimension lors de tous les scénarios, cela rend cette quantité difficile à être exactement évalué. On pourrait enlever cette quantité t et la remplacer par quelque chose qui dépend de n - le nombre de noeuds d'évacuation et m - la taille maximale d'un chemin

d'évacuation en remplaçant ce tableau de ressource avec une autre structure.

On a aussi pas encore implémenté la méthode d'évaluation des voisins en fonction de la liste d'évacuation initiale, qui aurait pu aider à réduire la complexité des algorithmes de recherche locale.

6 Conclusion

En conclusion, nous avons découvert de nouvelles méthodologies algorithmiques à travers un sujet de TP très réaliste. Ces compétences obtenues vont nous aider d'une part dans l'acquisition de nouvelles connaissances dans le domaine d'apprentissage et d'intélligence artificielle et d'autre part dans l'idéologie et la méthode de travail dans la vie professionnelle d'avenir. A l'issue de ce TP, on peut procéder non seulement aux améliorations du projet courrant mais aussi aux autres nouveaux et fascinants projets.

A Annexe : Tableaux des résultats

File_name	Graph_length	Borne_Inf	Execution_time
dense_10_30_3_1_I	126	91	0.006140470504760742
dense_10_30_3_2_I	140	120	0.006211996078491211
dense_10_30_3_3_I	120	106	0.007428884506225586
dense_10_30_3_4_I	81	124	0.005032062530517578
dense_10_30_3_5_I	102	117	0.005662679672241211
dense_10_30_3_6_I	113	126	0.0083770751953125
dense_10_30_3_7_I	54	104	0.002274036407470703
dense_10_30_3_8_I	95	105	0.006368398666381836
dense_10_30_3_9_I	88	98	0.005434513092041016
dense_10_30_3_10_I	96	101	0.005625009536743164
medium_10_30_3_1_I	131	106	0.005955934524536133
medium_10_30_3_2_I	95	96	0.003797769546508789
medium_10_30_3_3_I	66	114	0.0028486251831054688
medium_10_30_3_4_I	71	129	0.004012346267700195
medium_10_30_3_5_I	89	124	0.0046460628509521484
medium_10_30_3_6_I	91	138	0.0055713653564453125
medium_10_30_3_7_I	65	103	0.003046751022338867
medium_10_30_3_8_I	101	96	0.004671335220336914
medium_10_30_3_9_I	94	111	0.004198312759399414
medium_10_30_3_10_I	91	98	0.0043642520904541016
sparse_10_30_3_1_I	85	111	0.0029268264770507812
sparse_10_30_3_2_I	71	108	0.0018994808197021484
sparse_10_30_3_3_I	80	110	0.0027391910552978516
sparse_10_30_3_4_I	79	128	0.004259347915649414
sparse_10_30_3_5_I	79	131	0.003480672836303711
sparse_10_30_3_6_I	66	108	0.0029387474060058594
sparse_10_30_3_7_I	54	121	0.002077341079711914
sparse_10_30_3_8_I	67	107	0.0026612281799316406
sparse_10_30_3_9_I	83	111	0.00287628173828125
sparse_10_30_3_10_I	50	100	0.0017762184143066406

Table 1: Tableau de résultats du calcul de borne inférieure

File_name	Graph_length	Borne_Sup	Execution_time
dense_10_30_3_1_I	126	184	0.188523530960083
dense_10_30_3_2_I	140	294	0.44780397415161133
dense_10_30_3_3_I	120	422	0.7390239238739014
dense_10_30_3_4_I	81	397	1.0182442665100098
dense_10_30_3_5_I	102	419	0.42987799644470215
dense_10_30_3_6_I	113	215	0.2892310619354248
dense_10_30_3_7_I	54	431	0.28623366355895996
dense_10_30_3_8_I	95	350	0.7839038372039795
dense_10_30_3_9_I	88	401	0.6881287097930908
dense_10_30_3_10_I	96	398	0.372039794921875
medium_10_30_3_1_I	131	288	0.2583346366882324
medium_10_30_3_2_I	95	382	0.19846677780151367
medium_10_30_3_3_I	66	357	0.2094438076019287
medium_10_30_3_4_I	71	410	0.4109303951263428
medium_10_30_3_5_I	89	407	0.3021891117095947
medium_10_30_3_6_I	91	283	0.29418373107910156
medium_10_30_3_7_I	65	432	0.5425488948822021
medium_10_30_3_8_I	101	292	0.9175183773040771
medium_10_30_3_9_I	94	366	0.5695192813873291
medium_10_30_3_10_I	91	398	0.6156201362609863
sparse_10_30_3_1_I	85	161	0.3267993927001953
sparse_10_30_3_2_I	71	202	0.13169527053833008
sparse_10_30_3_3_I	80	288	0.13466429710388184
sparse_10_30_3_4_I	79	408	0.7890996932983398
sparse_10_30_3_5_I	79	257	0.22382283210754395
sparse_10_30_3_6_I	66	216	0.18708562850952148
sparse_10_30_3_7_I	54	228	0.14113593101501465
sparse_10_30_3_8_I	67	237	0.1445012092590332
sparse_10_30_3_9_I	83	338	0.14860248565673828
sparse_10_30_3_10_I	50	201	0.14015889167785645

Table 2: Tableau de résultats du calcul de borne supérieure

File_name	Graph_length	Borne_Sup	Result	Difference	Execution_time
dense_10_30_3_1_I	126	184	180	4	31.92162036895752
dense_10_30_3_2_I	140	294	288	6	70.95427417755127
dense_10_30_3_3_I	120	422	421	1	201.69897150993347
dense_10_30_3_4_I	81	397	388	9	257.1024286746979
dense_10_30_3_5_I	102	419	403	16	120.86313343048096
dense_10_30_3_6_I	113	215	213	2	53.73650622367859
dense_10_30_3_7_I	54	431	420	11	55.4623019695282
dense_10_30_3_8_I	95	350	341	9	185.69349265098572
dense_10_30_3_9_I	88	401	394	7	146.73166918754578
dense_10_30_3_10_I	96	398	389	9	101.84446477890015
medium_10_30_3_1_I	131	288	287	1	51.70178818702698
medium_10_30_3_2_I	95	382	371	11	30.989169120788574
medium_10_30_3_3_I	66	357	350	7	60.86128854751587
medium_10_30_3_4_I	71	410	393	17	135.02099466323853
medium_10_30_3_5_I	89	407	405	2	88.07231855392456
medium_10_30_3_6_I	91	283	283	0	34.093809843063354
medium_10_30_3_7_I	65	432	426	6	59.863933801651
medium_10_30_3_8_I	101	292	290	2	106.81437015533447
medium_10_30_3_9_I	94	366	362	4	117.48781752586365
medium_10_30_3_10_I	91	398	398	0	109.34374022483826
sparse_10_30_3_1_I	85	161	145	16	25.378117084503174
sparse_10_30_3_2_I	71	202	183	19	10.543813705444336
sparse_10_30_3_3_I	80	288	272	16	36.58016514778137
sparse_10_30_3_4_I	79	408	388	20	189.3014805316925
sparse_10_30_3_5_I	79	257	228	29	12.405827760696411
sparse_10_30_3_6_I	66	216	211	5	28.690287351608276
sparse_10_30_3_7_I	54	228	224	4	9.71701717376709
sparse_10_30_3_8_I	67	237	227	10	28.95856761932373
sparse_10_30_3_9_I	83	338	322	16	37.43889784812927
sparse_10_30_3_10_I	50	201	188	13	25.069968700408936

Table 3: Tableau de résultats du calcul du temps total avec la méthode de recherche locale

File_name	Graph_length	LocalSearchIntensified_Result	Result	Difference	Execution_time
dense_10_30_3_1_I	126	180	180	0	86.22749733924866
dense_10_30_3_2_I	140	288	288	0	157.23371815681458
dense_10_30_3_3_I	120	421	421	0	565.6836860179901
dense_10_30_3_4_I	81	388	388	0	562.261132478714
dense_10_30_3_5_I	102	403	403	0	334.1206569671631
dense_10_30_3_6_I	113	213	213	0	157.75252199172974
dense_10_30_3_7_I	54	420	420	0	153.50326919555664
dense_10_30_3_8_I	95	341	341	0	522.8742191791534
dense_10_30_3_9_I	88	394	394	0	370.9824845790863
dense_10_30_3_10_I	96	389	389	0	310.33453726768494
medium_10_30_3_1_I	131	287	287	0	155.30231642723083
medium_10_30_3_2_I	95	371	371	0	70.63849830627441
medium_10_30_3_3_I	66	350	350	0	188.11520886421204
medium_10_30_3_4_I	71	393	393	0	361.40907311439514
medium_10_30_3_5_I	89	405	405	0	255.55352973937988
medium_10_30_3_6_I	91	283	283	0	138.876624584198
medium_10_30_3_7_I	65	426	426	0	200.00871014595032
medium_10_30_3_8_I	101	290	290	0	301.2664408683777
medium_10_30_3_9_I	94	362	362	0	301.9045350551605
medium_10_30_3_10_I	91	398	398	0	342.5757882595062
sparse_10_30_3_1_I	85	145	137	8	85.7442786693573
sparse_10_30_3_2_I	71	183	183	0	36.555091381073
sparse_10_30_3_3_I	80	272	272	0	90.74098634719849
sparse_10_30_3_4_I	79	388	388	0	488.35104489326477
sparse_10_30_3_5_I	79	228	223	5	40.97431540489197
sparse_10_30_3_6_I	66	211	211	0	81.29766273498535
sparse_10_30_3_7_I	54	224	224	0	48.49684548377991
sparse_10_30_3_8_I	67	227	227	0	77.89586400985718
sparse_10_30_3_9_I	83	322	322	0	114.04516339302063
sparse_10_30_3_10_I	50	188	188	0	85.93571448326111

Table 4: Tableau de résultats du calcul du temps total avec la méthode de recherche locale multi-start

Solution_name	AllStartAt0	BorneSupSolution	LocalSearchIntensified	LocalSearchDiversified
dense_10_30_3_1_I	Invalid	Valid	Valid	Valid
dense_10_30_3_2_I	Invalid	Valid	Valid	Valid
dense_10_30_3_3_I	Invalid	Valid	Valid	Valid
dense_10_30_3_4_I	Invalid	Valid	Valid	Valid
dense_10_30_3_5_I	Invalid	Valid	Valid	Valid
dense_10_30_3_6_I	Invalid	Valid	Valid	Valid
dense_10_30_3_7_I	Invalid	Valid	Valid	Valid
dense_10_30_3_8_I	Invalid	Valid	Valid	Valid
dense_10_30_3_9_I	Invalid	Valid	Valid	Valid
dense_10_30_3_10_I	Invalid	Valid	Valid	Valid
medium_10_30_3_1_I	Invalid	Valid	Valid	Valid
medium_10_30_3_2_I	Invalid	Valid	Valid	Valid
medium_10_30_3_3_I	Invalid	Valid	Valid	Valid
medium_10_30_3_4_I	Invalid	Valid	Valid	Valid
medium_10_30_3_5_I	Invalid	Valid	Valid	Valid
medium_10_30_3_6_I	Invalid	Valid	Valid	Valid
medium_10_30_3_7_I	Invalid	Valid	Valid	Valid
medium_10_30_3_8_I	Invalid	Valid	Valid	Valid
medium_10_30_3_9_I	Invalid	Valid	Valid	Valid
medium_10_30_3_10_I	Invalid	Valid	Valid	Valid
sparse_10_30_3_1_I	Invalid	Valid	Valid	Valid
sparse_10_30_3_2_I	Invalid	Valid	Valid	Valid
sparse_10_30_3_3_I	Invalid	Valid	Valid	Valid
sparse_10_30_3_4_I	Invalid	Valid	Valid	Valid
sparse_10_30_3_5_I	Invalid	Valid	Valid	Valid
sparse_10_30_3_6_I	Invalid	Valid	Valid	Valid
sparse_10_30_3_7_I	Invalid	Valid	Valid	Valid
sparse_10_30_3_8_I	Invalid	Valid	Valid	Valid
sparse_10_30_3_9_I	Invalid	Valid	Valid	Valid
sparse_10_30_3_10_I	Invalid	Valid	Valid	Valid

Table 5: Tableau de résultat d'application du vérificateur de solutions sur les solutions obtenues

References

[1] M. J. Huget, "Tp métaheuristiques- 4e ir insa," 2018. [Online]. Available: https://homepages.laas.fr/huguet/drupal/sites/homepages.laas.fr.huguet/files/u78/2018-2019-TP-meta.pdf

[2] C. Artigues, E. Hébrard, Y. Pencolé, A. Schutt, and P. J. Stuckey, "Data instance generator and optimization models for evacuation planning in the event of wildfire," 2018. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01814063/document