

Techniques Avancées d'IA

Méthodes incomplètes d'optimisation combinatoire

On n'a pas d'algorithme permettant de trouver "directement" (efficacement) et à tous les coups la solution optimale de nombreux problèmes d'optimisation combinatoire.

Mais on connaît des algorithmes "gloutons".

⇒ On peut essayer d'améliorer la solution gloutonne, par succession de petites modifications.

Par exemples, pour le problème du sac-à-dos :

- échanger deux objets (un dans le sac, l'autre en dehors)
- ajouter un objet (au cas où un échange aurait fait suffisamment de place)

Algorithme générique de recherche locale

Algorithme d'exploration de base :

(où $S \succ S'$ signifie : S est meilleure que S')

(a) $S \leftarrow$ une solution initiale ; fini \leftarrow faux ;

(b) tant que *pas fini* faire :

$S' \leftarrow$ choisir_un_voisin(S) ;

 si $S' \succ S$ alors $S \leftarrow S'$;

 sinon fini \leftarrow vrai ;

(c) retourner S .

Algorithme générique de recherche locale

Algorithme d'exploration de base :

(où $S \succ S'$ signifie : S est meilleure que S')

(a) $S \leftarrow$ une solution initiale ; fini \leftarrow faux ;

(b) tant que *pas fini* faire :

$S' \leftarrow$ choisir_un_voisin(S) ;

si $S' \succ S$ alors $S \leftarrow S'$;

sinon fini \leftarrow vrai ;

(c) retourner S .

Choix d'un voisin :

- Un *voisin* de la solution courante S est une solution S' obtenue à partir de S en appliquant une “petite” modification.
- rapide : un algo qui calcule un seul voisin !
- plus long : on énumère tous les voisins, on garde le meilleur (*montée selon la plus grande pente*)
- intermédiaire : on génère un échantillon de voisins, on garde le meilleur

Algorithme générique de recherche locale

Exploration de base avec énumération du voisinage :

- (a) $S \leftarrow$ une solution initiale ; fini \leftarrow faux ;
- (b) tant que *pas fini* faire :
 - $S' \leftarrow S$;
 - pour tous les voisins \hat{S} de S faire : si $\hat{S} \succ S'$ alors $S' \leftarrow \hat{S}$;
 - si $S' \succ S$ alors $S \leftarrow S'$;
 - sinon fini \leftarrow vrai ;
- (c) retourner S .

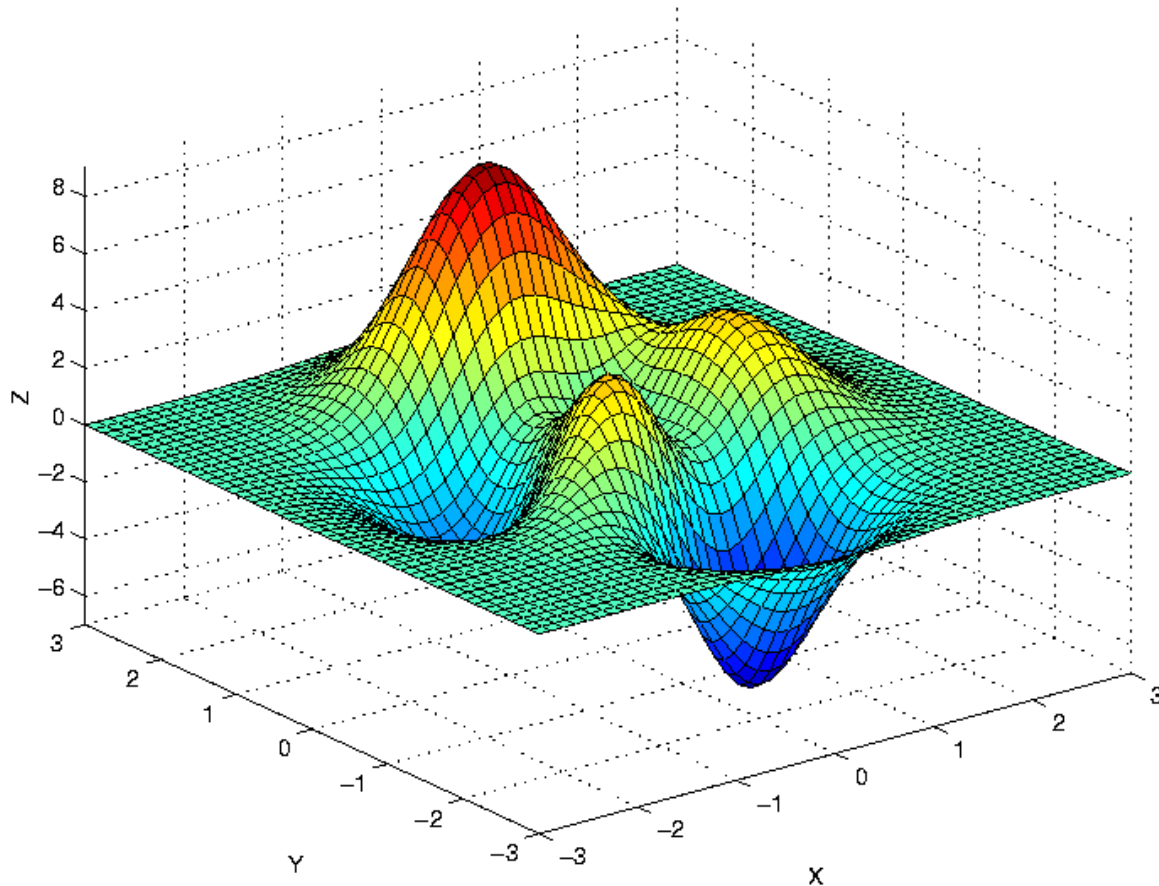
Exploration de base avec échantillonnage du voisinage :

- (a) $S \leftarrow$ une solution initiale ; fini \leftarrow faux ;
- (b) tant que *pas fini* faire :
 - $S' \leftarrow S$;
 - pour i allant de 1 à NbTirages faire :
 - tirer un voisin \hat{S} de S ; si $\hat{S} \succ S'$ alors $S' \leftarrow \hat{S}$;
 - si $S' \succ S$ alors $S \leftarrow S'$;
 - sinon fini \leftarrow vrai ;
- (c) retourner S .

Algorithme générique de recherche locale

Problème important : les optimums locaux

- moins de chance d'en rencontrer si voisinage “grand”
- mais grand voisinage = modifications complexes et plus de voisins



Algorithme générique de recherche locale

⇒ Algorithme d'exploration avec redémarrages :

1. $S^* \leftarrow$ une solution initiale ;
2. Tant que *on a du temps* faire :
 - (a) $S \leftarrow$ une nouvelle solution initiale ; fini \leftarrow faux ;
 - (b) Tant que *pas fini* faire :
 - $S' \leftarrow$ choisir_un_voisin(S) ;
 - si $S' \succ S$ alors $S \leftarrow S'$;
 - sinon fini \leftarrow vrai ;
 - (c) si $S \succ S^*$ alors $S^* \leftarrow S$;
3. retourner(S^*).

- À l'étape 2a : génération de solutions (partiellement) aléatoires
- ⇒ on ne tombe pas toujours sur les mêmes optimums locaux

Algorithme générique de recherche locale

Exploration avec redémarrage et voisinage exhaustif :

1. $S^* \leftarrow$ une solution initiale ;

2. Tant que *on a du temps* faire :

(a) $S \leftarrow$ une nouvelle solution initiale ; fini \leftarrow faux ;

(b) tant que *pas fini* faire :

$S' \leftarrow S$;

pour tous les voisins \hat{S} de S faire : si $\hat{S} \succ S'$ alors $S' \leftarrow \hat{S}$;

si $S' \succ S$ alors $S \leftarrow S'$;

sinon fini \leftarrow vrai ;

(c) si $S \succ S^*$ alors $S^* \leftarrow S$;

3. retourner(S^*).

Algorithme générique de recherche locale

⇒ **Mémoire à court terme : liste *tabou*** Pour sortir d'un optimum local : autoriser le passage par une solution un peu moins optimale.

Problème : on risque de retomber immédiatement après dans le même minimum local

⇒ on mémorise à l'aide d'une *liste tabou* les solutions récemment explorées, où ça n'est pas la peine de retourner.

Algorithme générique de recherche locale

1. $S^* \leftarrow$ une solution initiale ;
2. tant que *on a du temps* faire :
 - (a) $S \leftarrow$ une nouvelle solution initiale ; **tabous** \leftarrow [] ;
 - (b) ~~tant que pas fini~~ pour i allant de 1 à NbPas faire :
 $S' \leftarrow$ meilleur_voisin(S , **tabous**) ;
si tabous plein
supprimer de tabous la solution la plus ancienne ;
tabous \leftarrow **tabous** + S ;
~~si $S' \succ S$ alors~~ $S \leftarrow S'$;
 - (c) si $S \succ S^*$ alors $S^* \leftarrow S$; ~~sinon fini~~ \leftarrow ~~vrai~~ ;
3. retourner(S^*).

Condition d'arrêt : on ne s'arrête plus à un optimum local, donc on utilise par exemple un compteur, et on ne fait la boucle 2b qu'un nombre fixé de fois.

Algorithme générique de recherche locale

Exemple : capacité 12kg, objets ci-dessous :

numéro	0	1	2	3
poids	3	4	5	6
valeur	8	10	11	12

Init: $S = \{(5, 11), (6, 12)\} = \text{optimum local ;}$

Voisins : $\{(4, 10), (6, 12)\}; \{(3, 8), (6, 12)\}$
 $\{(4, 10), (5, 11)\}; \{(3, 8), (5, 11)\}$

Algorithme générique de recherche locale

Exemple : capacité 12kg, objets ci-dessous :

numéro	0	1	2	3
poids	3	4	5	6
valeur	8	10	11	12

Init: $S = \{(5, 11), (6, 12)\} = \text{optimum local ;}$

Voisins : $\{(4, 10), (6, 12)\}; \{(3, 8), (6, 12)\}$
 $\{(4, 10), (5, 11)\}; \{(3, 8), (5, 11)\}$

solution suivante = $\{(4, 10), (6, 12)\}$; solution initiale \rightarrow liste tabou ;

Algorithme générique de recherche locale

Exemple : capacité 12kg, objets ci-dessous :

numéro	0	1	2	3
poids	3	4	5	6
valeur	8	10	11	12

Init: $S = \{(5, 11), (6, 12)\} = \text{optimum local ;}$

Voisins : $\{(4, 10), (6, 12)\}; \{(3, 8), (6, 12)\}$
 $\{(4, 10), (5, 11)\}; \{(3, 8), (5, 11)\}$

solution suivante = $\{(4, 10), (6, 12)\}$; solution initiale \rightarrow liste tabou ;

solution suivante = $\{(4, 10), (5, 11)\}$;

Algorithme générique de recherche locale

Exemple : capacité 12kg, objets ci-dessous :

numéro	0	1	2	3
poids	3	4	5	6
valeur	8	10	11	12

Init: $S = \{(5, 11), (6, 12)\} = \text{optimum local ;}$

Voisins : $\{(4, 10), (6, 12)\}; \{(3, 8), (6, 12)\}$
 $\{(4, 10), (5, 11)\}; \{(3, 8), (5, 11)\}$

solution suivante = $\{(4, 10), (6, 12)\}$; solution initiale \rightarrow liste tabou ;

solution suivante = $\{(4, 10), (5, 11)\}$;

solution suivante par ajout de $(3, 8)$;

une taille de 1 a suffi pour sortir de l'optimum local.

Algorithme générique de recherche locale

Remarques :

- Liste tabou implémentée en FIFO
- Détermination de la taille de la liste par expérimentation. On peut utiliser une liste plus courte si on ne parcourt pas tout le voisinage mais seulement un échantillon de celui-ci (cela réduit le risque de cycle)
- Lorsque les problèmes sont décrites par bcp d'attributs, on ne stocke pas les solutions entières, mais certains attributs de ces solutions – ou des mouvements permettant d'arriver à ces solutions.

Mémoire à moyen terme

Intensification de la recherche lorsqu'on est dans une zone prometteuse.

On stocke pour chaque valeur de chaque / certains attribut des solutions le nombre de fois où il a été présent dans une solution récente ; on peut alors poursuivre la recherche pendant un moment en fixant certains attributs à des valeurs qui semblent apparaître souvent dans des bonnes solutions. (ce qui peut revenir à changer la structure du voisinage durant cette phase)

Mémoire à long terme

Diversification de la recherche.

Pour éviter d'ignorer certaines parties de l'espace de recherche.

On stocke pour chaque valeur de chaque certains attributs les fréquences d'apparition dans les solutions parcourues ; lorsqu'on génère une nouvelle solution, on force certaines valeurs peu apparues jusque là.

Autres améliorations possibles

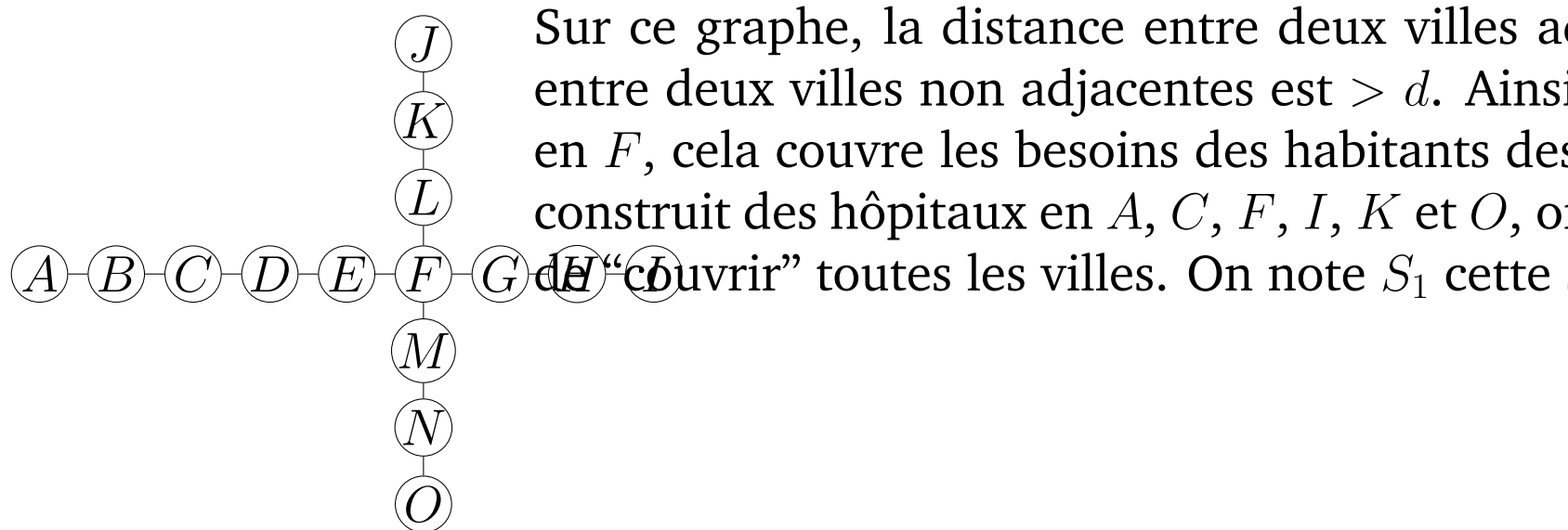
Aspiration On peut autoriser une solution $S' \succ S$, même si le mouvement est interdit par la liste tabou.

Fonction de coût/valeur auxiliaire Pour guider la recherche, notamment lorsque la fonction à optimiser ne prend que peu de valeurs. Par exemples, dans le problème des déménageurs, on peut chercher à maximiser $\sum_i^N V_i^\beta / N^\alpha$, avec $\alpha, \beta \geq 1$, ça favorise les répartitions plus homogènes.

Exercices

Exercice 1 On suppose qu'on doit choisir des villes où construire des hôpitaux, afin qu'aucune ville ne soit à une distance plus grande qu'un d fixé d'un hôpital. On cherche bien entendu à construire le moins d'hôpitaux possible.

Par exemple, on suppose qu'on a 15 villes A à O , disposées ainsi :



Question 1.1 Donnez une solution optimale sur cet exemple.

De manière générale, on suppose qu'on a n villes V_1, \dots, V_n , et qu'on connaît les distances entre les villes ; on note d_{ij} la distance entre V_i

Exercices

et V_j .

Question 1.2 Donnez un algorithme *glouton* simple pour résoudre approximativement le problème. Déroulez votre algorithme sur l'exemple ci-dessus, et indiquez la solution finale obtenue.

On veut maintenant résoudre ce problème à l'aide d'une recherche locale.

Question 1.3 Donnez une définition intéressante de la notion de voisinage pour ce problème. Peut-on arriver à la solution optimale à partir de S_1 avec une montée selon la plus grande pente avec votre notion de voisinage ?

Question 1.4 Que peut-on stocker dans une liste tabou pour passer les minimums locaux ? Quelle longueur minimale de liste tabou faudrait-il pour arriver à coup sûr à une solution optimale à partir de S_1 ?

Exercice 2 On a m camions, initialement garés dans m dépôts (qui contiennent les mêmes marchandises) et n clients à livrer. On connaît les distance entre les clients. Pour optimiser le coût des livraisons, il

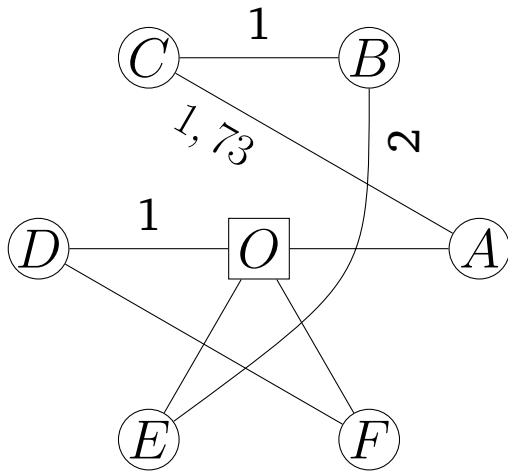
Exercices

s'agit de répartir, en les ordonnants, les livraisons entre les camions, en minimisant la distance parcourue par l'ensemble des camions – en comptant le retour de chaque camion à son dépôt initial.

On a m camions C_1, \dots, C_m qui partent tous de la même ville, et n clients à livrer dans n villes V_1, \dots, V_n . On suppose qu'on connaît les distances entre les villes. Pour optimiser le coût des livraisons, il s'agit de répartir, en les ordonnant, les livraisons entre les camions, en minimisant la distance parcourue par l'ensemble des camions – en comptant le retour de chaque camion à son dépôt initial.

Par exemple, on peut considérer deux camions, un dépôt O , 6 villes à F , disposées en hexagone régulier toutes à la distance 1 du dépôt : adjacentes sont à une distance 1, et deux villes X et Z situées de part d'une ville Y sont à une distance de $\sqrt{3} = 1,73$. Les arcs indiquent les deux camions dans une solution initiale $S_0 = \{(ACBE), (DF)\}$:

Exercices



Question 2.1 Qu'est-ce qui définit une solution du problème ?

Question 2.2 Donnez une définition intéressante de la notion de voisinage d'une solution.

Question 2.3 Donnez une solution optimale.

Question 2.4 Combien faut-il de mouvements pour arriver à un optimum (local ou global) avec votre notion de voisinage sur cet exemple ?

Question 2.5 Que peut-on stocker dans une liste tabou sur ce type de problème pour s'autoriser à sortir des optimums locaux ?

Exercices

Exercice 3 Pour construire un décor de théâtre, un charpentier a besoin de planches de même section, et de longueurs variables. Plus précisément, il a besoin des longueurs suivantes : 240, 240, 150, 90, 150, 150, 150, 180, 180, 240, 240, 120, 120, 60 centimètres. La menuiserie où il se fournit vend ces planches en longueur fixe : 360 centimètres. Il s'agit d'acheter le moins de planches possibles. (C'est un problème équivalent au problème des déménageurs).

Question 3.1 Donnez un algorithme glouton pour résoudre approximativement ce problème - quelle solution obtenez-vous.

Question 3.2 Pour une méthode de recherche locale, comment peut-on définir le voisinage d'une solution ?

Question 3.3 Si on ne veut pas stocker des solutions entières dans une liste tabou, quelles informations sur les solutions rencontrées peut-on y mémoriser ?

Autres méthodes incomplètes

- On maintient un ensemble de K solutions courantes.
- À chaque itérations, on sélectionne certaines de ces solutions courantes et on les modifie :
 - * par “croisement” / mélange de deux solutions courantes
 - * par “mutation” (cf voisinage)
- La sélection des solutions courantes à modifier se fait de manière aléatoire, en donnant une probabilité plus élevées aux meilleurs solutions courantes.

Autres méthodes incomplètes : Algorithmes génétiques

Un algorithme type :

1. $P \leftarrow$ population initiale = K solutions initiales ;
2. Tant qu'on a du temps, faire :
 - (a) $C \leftarrow \{\}$; $P_1 \leftarrow \text{select}(P)$; $P \leftarrow P - P_1$;
 - (b) Tant que $|P_1| \geq 2$ faire :
 - i. $\text{parent}_1, \text{parent}_2 \leftarrow 2$ éléments distincts de P_1 ;
 $P_1 \leftarrow P_1 - \{\text{parent}_1, \text{parent}_2\}$;
 - ii. $\text{enfant}_1, \text{enfant}_2 \leftarrow \text{croisement}(\text{parent}_1, \text{parent}_2)$;
 - iii. $C \leftarrow C \cup \{\text{mutation}(\text{enfant}_1), \text{mutation}(\text{enfant}_2)\}$;
 - (c) $P \leftarrow P \cup C$;

Problème : dans certaines applications, définir des opérations de croisement / mutation qui retournent des solutions possibles n'est pas évident.

Autres méthodes incomplètes : Recherche informée incomplète

On fait une recherche informée, type A^* , mais on élague la recherche.

Exemple sur le problème du routage / voyageur de commerce :

- un état de la recherche est une séquence (partielle) de clients à visiter ;
- l'état initial est la séquence vide ;
- à partir d'un état $[c_1, \dots, c_k]$, on produit toutes les séquences obtenues en ajoutant un client pas déjà présent dans la séquences ;
- la “valeur” d'un état est le trajet “déjà parcouru”.

Recherche type A^* :

- on remplace la séquence $[c_1, \dots, c_k]$ par tous ses successeurs
- à chaque itération, on développe la séquence qui a le coût courant le plus faible

Remarque : l'heuristique (coût déjà parcouru) est minorante, donc certitude de trouver la route la plus courte.

Problème : la taille de la file d'attente !

⇒ recherche incomplète : on ne garde qu'une partie de la file d'attente.

Autres méthodes incomplètes : Recherche informée incomplète

Recherche “en faisceau” :

- la taille de la file d’attente est fixée à un T donné
- on fait une recherche *en largeur d’abord* :
 1. générer tous les successeurs possibles de tous les états de la file d’attente
 2. on ne remet dans la file d’attente que les T successeurs les meilleurs (coût minimal)

Exercices

Exercice 4 Définir des opérations de mutation et de croisement qui permettraient de résoudre les problèmes suivants avec un algorithme génétique : le compte est bon, le sac-à-dos, le voyageur de commerce (routage avec un seul camion), les déménageurs.

Projet : routage de véhicules avec fenêtres temporelles

Un problème important consiste à construire à définir les tournées de véhicules devant passer en un certains nombres de points avec des contraintes temporelles : pour chaque point qui doit être servi par un véhicule, on a deux contraintes définissant une heure minimale de passage et une heure maximale de livraison. On a aussi des contraintes de capacité : il faut livrer à chaque point des marchandises ayant un certain poids, et chaque véhicule a une contrainte sur le poids total qu'il peut emmener. Le fichier `routage_rc201.txt` contient une ligne par client, de la forme :

Id, X, Y, Quantité, Tmin, Tmax, Durée

où Id est un numéro permettant d'identifier les clients, et Durée indique le temps d'arrêt minimal chez le client (temps nécessaire à la livraison). Ce fichier généré par Marius Solomon se trouve, avec d'autres, à l'adresse <http://web.cba.neu.edu/~msolomon/problems.htm>. On suppose qu'on a un nombre illimité de camions disponibles, tous de même capacité. La première ligne donne la capacité des camions, et la seconde ligne donne la position du dépôt (point de départ et

Projet : routage de véhicules avec fenêtres temporelles

d'arrivée des véhicule) ainsi que l'heure maximale de retour.

Il s'agit de minimiser le temps total de parcours – la somme des temps de parcours de tous les camions. Pour simplifier, on assimilera le temps de parcours entre deux clients à la distance euclidienne entre ces deux clients : $\sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$.

Recherche de routes optimales sans contraintes On vous demande pour commencer d'implémenter un algorithme de recherche local simple – type montée selon la plus grande pente – avec redémarrages, mais sans mémoire. Il vous faudra donc pour commencer :

1. définir un type de données pour représenter les solutions ;
2. implémenter un algorithme de génération de solutions initiales aléatoires ;
3. pour “voir” les solutions, ça serait bien d'avoir un affichage des trajets sur une carte où on afficherait les positions des clients données par leurs coordonnées.

Une notion cruciale pour les recherches locales est la notion de voisinage. Il faut pouvoir calculer rapidement les voisins d'une solution

Projet : routage de véhicules avec fenêtres temporelles

courante. Vous devez donc réfléchir à la notion de voisinage, et implémenter des fonctions :

- permettant d'énumérer tous les voisins d'une solution donnée ;
- permettant de tirer au hasard un nombre fixé de voisins d'une solution donnée ;

Il ne reste alors plus qu'à implémenter l'algorithme de montée selon la plus grande pente.

Utilisation d'une liste tabou Dans bien des applications, une telle liste permet des améliorations importantes des temps de calcul et des solutions trouvées. Pour implémenter une recherche avec liste tabou, vous devez d'abord réfléchir à ce que vous pourrez mémoriser dans une telle liste – il est en général peu efficace de mémoriser des solutions récentes en entier, il vaut en général mieux mémoriser certaines caractéristiques de ces solutions, faciles à tester sur les nouvelles solutions générées. Quelle caractéristiques pourrait-on mémoriser ici ?

Projet : routage de véhicules avec fenêtres temporelles

Prise en compte des contraintes Implémentez finalement une recherche locale prenant en compte les contraintes de temps et de capacité : un camion ne doit pas transporter plus que sa capacité maximale ; si on arrive trop tôt chez un client, il faut attendre ; le temps d'arriver doit être suffisamment tôt pour que la fin de la livraison soit avant l'heure maximum de livraison.