TIPE : Le placement d'hôpitaux sur une carte de densité à l'aide d'optimisation combinatoire et les métaheuristiques

Mathieu Maxence

n°*SCEI* : 56639

Juin 2022

Désert médical

Définition 1.1

Un **désert médical** est un territoire caractérisé par la très grande difficulté pour les patients à accéder aux professionnels de santé.

Raisons des déserts médicaux

Raisons pour lesquelles ces déserts existent :

- Diminution du nombre de médecins
- Augementation du nombre de patients à traiter
- **Manque d'établissements de santé dans les régions rurales**
- **Saturation des établissements de santé**

Création d'hôpitaux

- Le plan *Ma Santé 2022* promettait la création de 1000 hôpitaux
- Devait permettre de mailler le territoire d'ici Juillet 2021

Problèmatique

Comment placer ces hôpitaux sur le territoire pour réduire le temps d'accès à un établissement de santé ?

Problèmatique

Comment placer ces hôpitaux sur le territoire pour réduire le temps d'accès à un établissement de santé ?

- Quelle modélisation peut-on faire d'un territoire pour traiter ce problème ?
- Comment évaluer la qualité d'une configuration ?
- Quelles méthodes permettent de trouver une configuration optimale en un minimum de temps ?

Plan

Première modélisation

Quelles caractéristiques du territoires sont à retenir dans la modélisation ?

- **La densité de population du territoire**
- **La position des hôpitaux**
- On néglige pour l'instant l'influence des routes et transports en commun.

Modélisation de la densité de population

On modélise le territoire par une matrice carrée de taille N.

Notation

Dans la suite, on appellera N le paramètre de taille.

Cette matrice décompose le territoire en N^2 sous-régions, caractérisées par le nombre de personnes vivant à l'interieur.

Modélisation de la densité de population

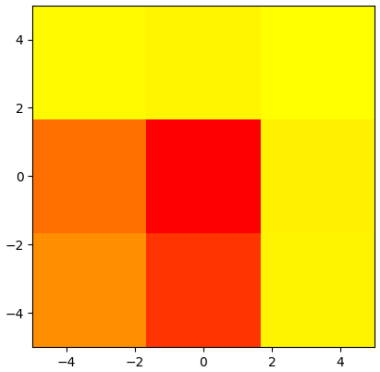


Figure: Carte de densité de population pour un paramètre N = 3. Plus la couleur est chaude, plus la population est dense.

$$\begin{bmatrix} 243 & 461 & 46 \\ 5690 & 10020 & 602 \\ 4454 & 7979 & 505 \end{bmatrix}$$

Figure: Valeurs de la matrice de densité de population

Analogie avec l'échantillonnage

On remarque que plus N est grand, plus la modélisation s'approche d'un modèle continu, en mesurant à des intervalles plus régulières la densité de population.

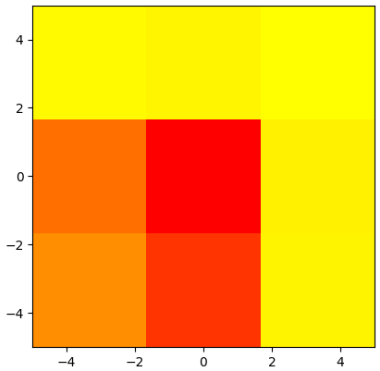


Figure: Carte de densité de population pour un paramètre N = 3

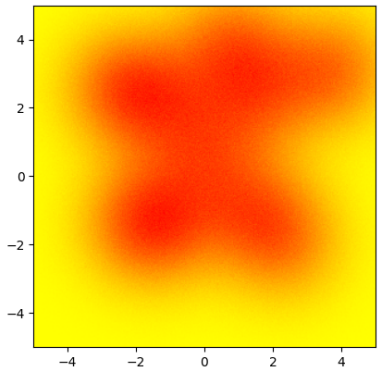


Figure: Carte de densité de population pour un paramètre N = 300

Analogie avec l'échantillonnage

Remarque

N est équivalent à une fréquence d'échantillonnage spatiale en m^{-1}.

Modélisation des hôpitaux

On représente les hôpitaux par un couple $(x, y) \in [\![0; N-1]\!]^2$.

Notation

Dans la suite, on appellera **solution** ou **configuration** un ensemble $S(x, y) \in [\![0; N-1]\!]^M$, avec M le nombre d'hôpitaux à placer.

Modélisation des hôpitaux

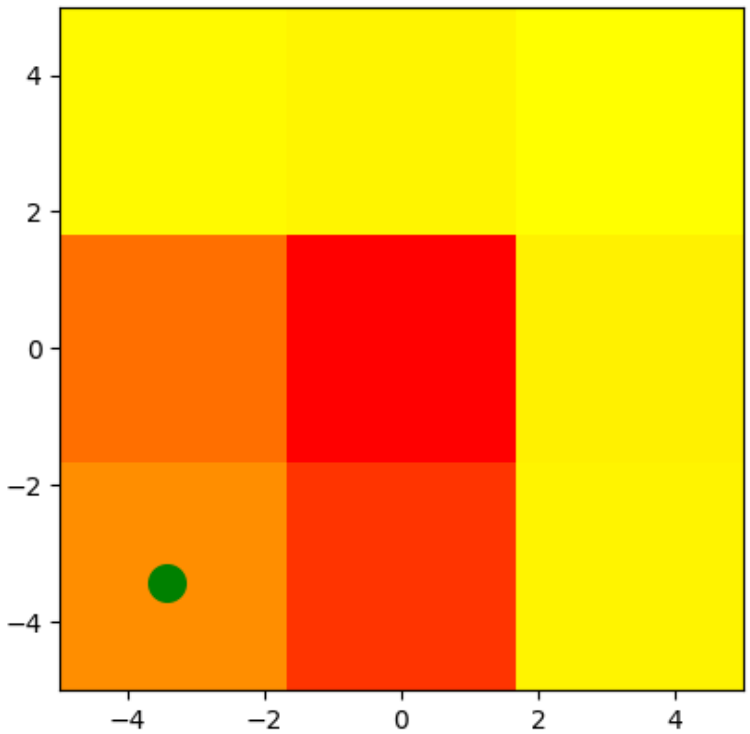


Figure: Carte de densité de population avec un hôpital aux coordonnées $(0, 2)$

Création d'une matrice de densité

Deux moyens pour obtenir une matrice de densité :

- Reproduire une carte de densité existante
- Générer aléatoirement une carte de densité

Observation pour la création d'une matrice de densité

Observation pour la création de matrice de densité

La population s'articule radialement autour de plusieurs "points chauds", qui sont généralement les grandes villes.
Cette hypothèse est particulèrement vraie à grande échelle.

Hypothèse pour la création d'une matrice de densité

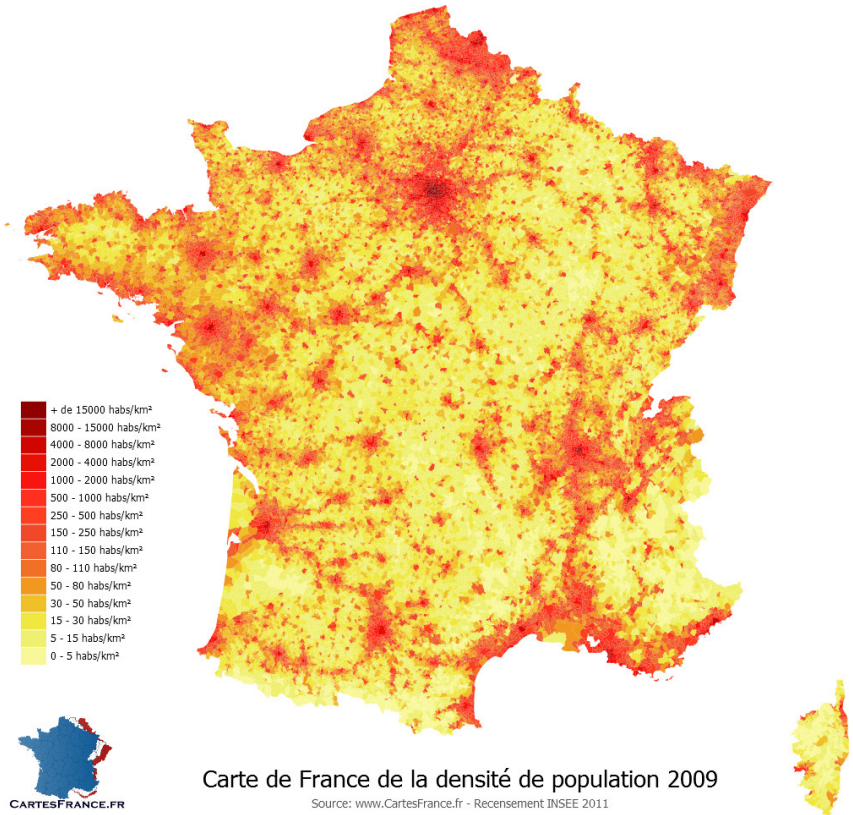


Figure: Carte de densité de la France. On remarque que la densité est particulièrement forte autour des grandes villes.

Méthode pour la création d'une matrice de densité

1. Générer aléatoirement un nombre fixe de coordonnées, qui représentent les grandes villes du territoire.
2. Pour chaque grande ville, remplir radialement les sous-regions autour de ces grandes villes : plus la sous-region est éloignée de la grande ville, moins elle est peuplée.

Exemple de la création d'une matrice de densité

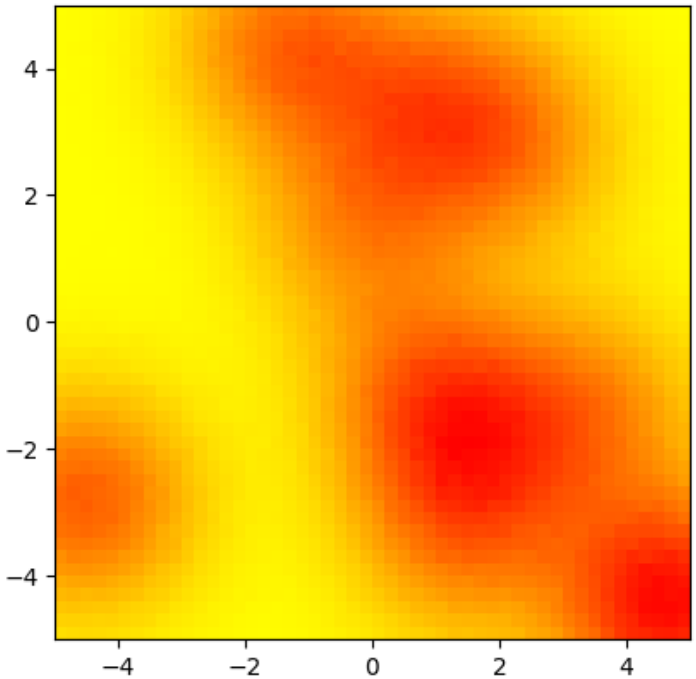


Figure: Carte de densité générée aléatoirement, avec N = 50

Limites de la génération d'une matrice de densité

Cette méthode possède plusieurs limitations :

- La possibilité de créer des "ilots solitaires"
- La répartition de la population est strictement radiale

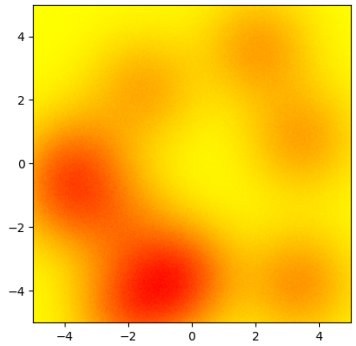


Figure: Exemple de génération avec des ilots

Fonction d'évaluation

On souhaite pouvoir comparer des configurations entre elles, pour déterminer quelle est la meilleure

Définition 2.1

Une fonction d'évaluation, notée f, associe une configuration à un score. Formellement, $f : [\![0; N-1]\!]^{2M} \to \mathbb{R}$.

Critères de la fonction d'évaluation

Dans certains problèmes, la fonction d'évaluation est évidente :

- Problème du voyageur de commerce $\rightarrow$ La distance du trajet
- Problème du sac à dos $\rightarrow$ La valeur du contenu du sac à dos

Dans la répartition d'hôpitaux, la qualité d'une configuration n'est pas évidente :

- La distance moyenne à un hôpital ?
- La distance médiane à un hôpital ?
- La pire distance ?

Critères de la fonction d'évaluation

On retiendra les critères suivants :

- La distance moyenne à un hôpital
- Le 95^{eme} percentile de la distance à un hôpital
- La distance médiane à un hôpital
- Le 5^{eme} percentile de la distance à un hôpital
- La pire distance à un hôpital
- La pire charge d'un hôpital

Définition 2.2

La charge d'un hôpital est le nombre de personnes rattachées à cet hôpital (la plus proche de son domicile).

Critères de la fonction d'évaluation

Remarque importante

Le choix des critères de la fonction d'évaluation est arbitraire et dépend entièrement du problème. On aurait pu choisir d'autres critères.

Exemple de la fonction d'évaluation

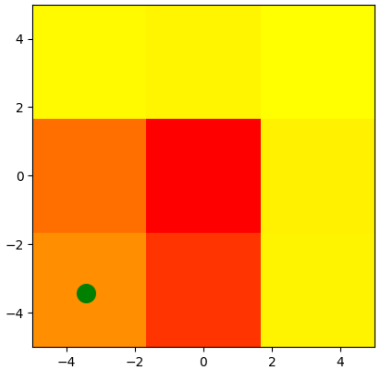


Figure: Carte de densité de population avec un hôpital aux coordonnées $(0, 2)$

- Distance moyenne : 58018
- 95^{eme} percentile : 1533
- Distance médiane : 14189
- 5^{eme} percentile : 188066
- Pire distance : 188066
- Pire charge : 30000

Coefficients de la fonction d'évaluation

Empiriquement, on décide d'appliquer les coefficients suivants aux critères de la fonction d'évaluation.

- Distance moyenne : 2
- 95^{eme} percentile : 3
- Distance médiane : 2
- 5^{eme} percentile : 1
- Pire distance : 1
- Pire charge : 10

Expression formelle du problème

Expression du problème d'optimisation combinatoire

- Soit H une matrice de densité telle que $H \in M_n(\mathbb{N})$.
- Soit f une fonction d'évaluation telle que $f : [\![0; N-1]\!]^{2M}. \to \mathbb{R}$

On cherche à déterminer $min\{f(S) : S \in [\![0; N-1]\!]^{2M}\}$

Dénombrement des solutions

1. On choisit un premier point pour placer le premier hôpital : N^2 choix.
2. On choisit un second point pour placer le second hôpital à un endroit différent de la première : $N^2 - 1$ choix.
3. On réalise cette opération M fois.

Résultat 3.1

En notant $S_{N,M}$ l'ensemble des solutions possibles pour une matrice de taille N pour M hôpitaux, on trouve que $Card(S_{N,M}) = \frac{(N^2)!}{(N^2-M)!}$

L'impossibilité du dénombrement des solutions

Exemple

En prenant N = 50, M = 10, $Card(S_{N,M})$ est de l'ordre de 10^{34}

Si on fait l'approximation que la fonction d'évaluation s'execute toujours en 1 μs, le temps requis pour calculer toutes les solutions s'élève à 10^{28} secondes, soit 10^{11} fois l'espérance de vie estimée de l'univers.

L'impossiblité d'une méthode exacte

- La résolution exacte par dénombrement est impossible car trop coûteuse en temps et en mémoire.
- La résolution exacte par une méthode "diviser pour régner" est aussi impossible.

Trouver la solution exacte du problème semble alors trop coûteux en temps, mais on souhaite tout de même trouver une solution satisfaisante en temps fini.

Les heuristiques

Définition 3.2

Une heuristique est une technique conçue pour résoudre un problème plus rapidement lorsque les méthodes classiques sont trop lentes, ou pour trouver une solution approximative lorsque les méthodes classiques ne parviennent pas à trouver une solution exacte.

Génération du voisinage

Définition 3.3

Le voisinage d'une solution est un sous-ensemble de solutions qu'il est possible d'atteindre par une série de transformations élémentaires.

Définition 3.4

L'ordre d'un voisinage est le nombre de transformations élémentaires possibles pour arriver aux solutions du voisinage.

Exemple de voisinage

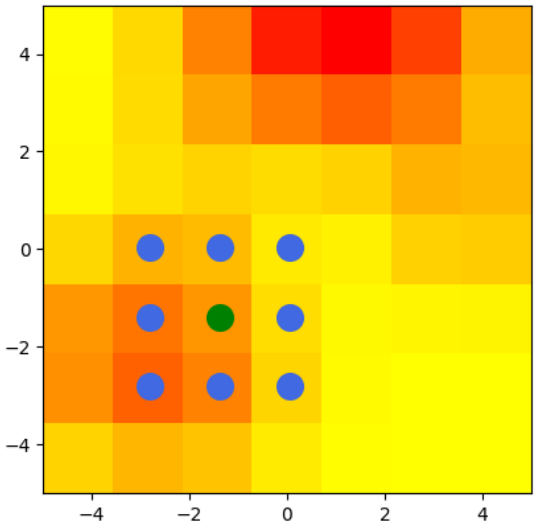


Figure: Exemple de voisinage d'ordre 1

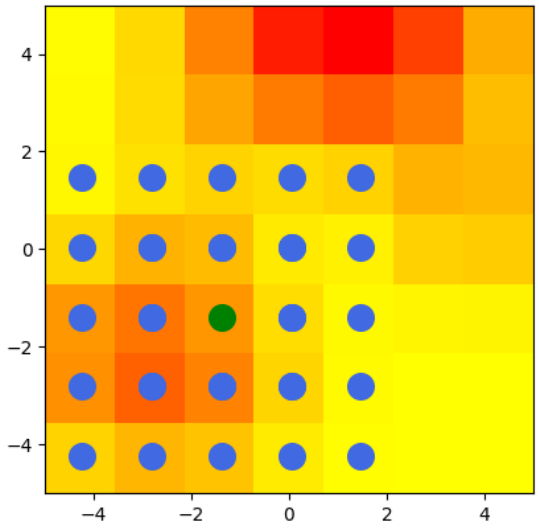


Figure: Exemple de voisinage d'ordre 2

Algorithme du gradiant

$N \leftarrow 1000$
$S \leftarrow$ *Générer une solution initiale aléatoire*
$i \leftarrow 0$
for $i \leq N$ **do**
    $V \leftarrow$ *Générer le voisinage de* S
    $S \leftarrow min\{f(S') : S' \in V\}$
    $i \leftarrow i + 1$
end for
return S

Importance du voisinage

- Si l'ordre du voisinage est trop grand, les temps de calculs deviennent très grands, et on retombe dans le problème initial.
- Si l'ordre du voisinage est trop petit, la solution tombe très rapidement dans un minimum local, où chaque solution du voisinage est pire que la solution actuelle.

Observation

Empiriquement, j'ai trouvé qu'un voisinage d'ordre 5 offrait un bon compromis entre temps d'exécution et qualité de la solution.

Algorithme glouton

Définiton 3.5

Un algorithme glouton est un algorithme du gradient avec un voisinage d'ordre 1.

Avantages et inconvénients des heuristiques

- Les heuristiques parviennent toujours à atteindre un minimum local, quelque soit sa qualité.
- Les heuristiques sont très rapides, et atteignent le minimum en très peu d'itérations.

Avantages et inconvénients des heuristiques

- Les heuristiques ne trouvent que très rarement le minimum global du problème.
- Les heuristiques n'ont aucun moyen de sortir d'un minimum local car leur voisinage est souvent assez faible.

Les méta-heuristiques

Définition 3.6

Les méta-heuristiques sont des heuristiques qui ont la capacité de sortir d'un minimum local grâce à un voisinage dynamique.

Algorithme du recuit simulé

Défintion 3.6

Le recuit simulé est une méta-heuristique qui s'inspire de la thermostatistique et de la métallurgie pour s'échapper des minimas locaux.

Analogie avec la thermostatistique

On peut s'inspirer du refroidissement du métal pour résoudre notre problème, en introduisant un nouveau paramètre : la **température**.

- Lorsque la température est élevée, le voisinage a un ordre élevé.
- Lorsque la température est faible, le voisinage a un ordre faible.

Remarque importante

Grace à la température, le voisinage est **dynamique**, c'est la force des méta-heuristiques.

Principe de l'algorithme du recuit simulé

Soit S' une solution aléatoire du voisinage de S.

1. On note $\Delta S = f(S') - f(S)$
2. On introduit le poids de Boltzman : $P(S') = P_0 e^{\frac{-\Delta S}{k_b T}}$
3. On génère un nombre aléatoire $R \in [0; 1]$
4. Si $R \leq P(S')$, on adopte S' comme la solution courante

Plus la température est haute, plus la probabilité qu'une configuration soit acceptée est haute

Algorithme du recuit simulé

$N \leftarrow 1000$
$S \leftarrow$ *Générer une solution initiale aléatoire*
$i \leftarrow 0$
for $i \leq N$ **do**
    $V \leftarrow$ *Générer le voisinage de* S
    $S' \leftarrow$ *Un élément de* V
    $\Delta S = f(S') - f(S)$
    if $\Delta S \geq 0$ **then**
        $S \leftarrow S'$
    else
        $P(S') \leftarrow P_0 e^{\frac{-\Delta S}{k_b T}}$
        $R \leftarrow$ *Réel aléatoire dans* $[0; 1]$
        if $P(S') \geq R$ **then**
            $S \leftarrow S'$
        end if
    end if
    $i \leftarrow i + 1$
    Diminuer la température T
end for
return S

Avantages et inconvénients du recuit simulé

- Donne de meilleurs résultats asymptotiquements
- Permet de sortir théoriquement de n'importe quel minimum local

Avantages et inconvénients du recuit simulé

- Introduction de deux nouveaux paramètres à gérer : P_0 et T
- Prend plus de temps à s'éxecuter

Observation sur les valeurs des paramètres

Empiriquement, $P_0 = 1$ et $T \in [0; 5000]$ donnent de bons résultats

Comparaison entre l'Algorithme Glouton et l'algorithme du recuit simulé

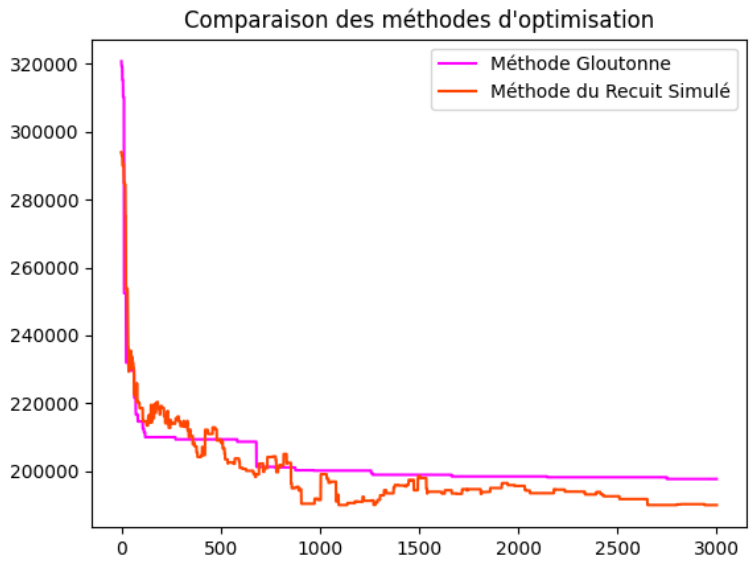


Figure: Exemple d'execution des deux algorithmes. L'abscisse est l'indice d'itération et l'ordonnée est le valeur de la fonction d'évaluation prise en la solution courante

Comparaison entre l'algorithme glouton et l'algorithme du recuit simulé

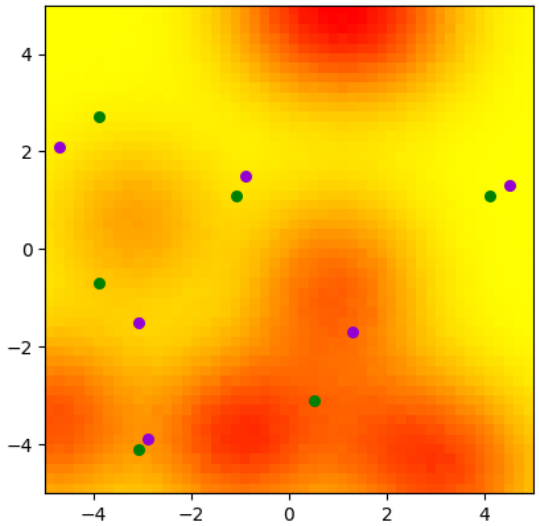


Figure: Exemple d'exécution des deux algorithmes sur une matrice de taille 50 avec 6 hôpitaux. Les points verts sont la solution de l'algorithme glouton, les points violets sont la solution de l'algorithme du recuit simulé.

Atteindre le minimum global

On couple l'algorithme du recuit simulé avec l'algorithme glouton à très faible voisinage :

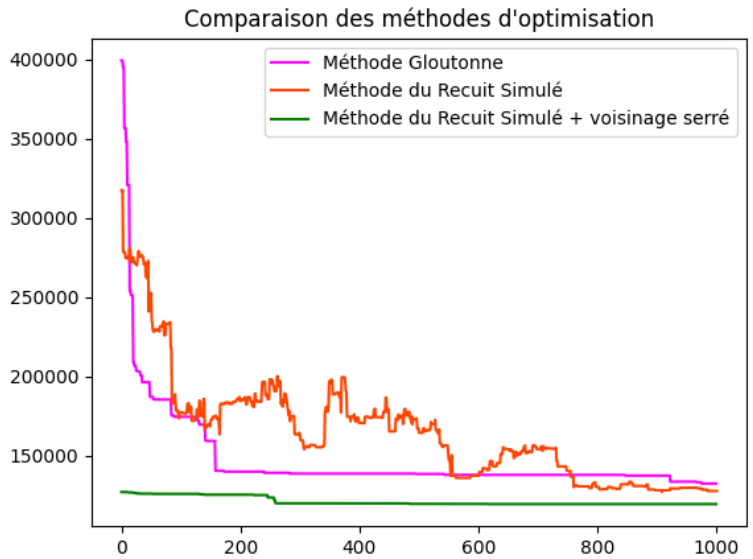


Figure: Exemple d'execution des deux algorithmes. L'abscisse est l'indice d'itération et l'ordonnée est la valeur de la fonction d'évaluation prise en la solution courante.

Les algorithmes génétiques

Les algorithmes génétiques utilisent la notion de sélection naturelle et l'appliquent à une population de solutions potentielles au problème donné

Observation pour la création d'une population de solution

On peut considérer une solution comme un être vivant, dont les gènes seraient les valeurs prises par la solution.

Pricipaux méchanismes génétiques

- La séléction naturelle : la moitié des gènes d'un nouvel individu correspond à ceux d'un des deux parents, l'autre moitié correspond à ceux de l'autre parent, et seuls les meilleurs individus sont conservés
- La dérive génétique : une partie des gènes de l'individu mute de manière aléatoire.

Principe de l'algorithme génétique

A partir d'une population initiale P

1. Créer un certain nombre de descendents
2. Les ajouter à la population
3. Faire muter la population
4. Conserver uniquement les meilleurs éléments de la population

Paramètres de l'algorithme génétique

- Le taux de mutation λ
- Le nombre de descendents R
- La taille de la population T

Algorithme génétique

$N \leftarrow 1000$
$\lambda \leftarrow 0.1$
$R \leftarrow 30$
$T \leftarrow 100$
$P \leftarrow$ *Générer T solutions initiales aléatoires*
$i \leftarrow 0$
for $i \leq N$ **do**
    $F \leftarrow$ *Generer R descendents de P*
    $P \leftarrow P + F$
    $P \leftarrow \lambda$ *mutations de P*
    $P \leftarrow$ *Les T meilleurs éléments de P*
end for
return *Meilleur élément de P*

Comparaisons des algorithmes

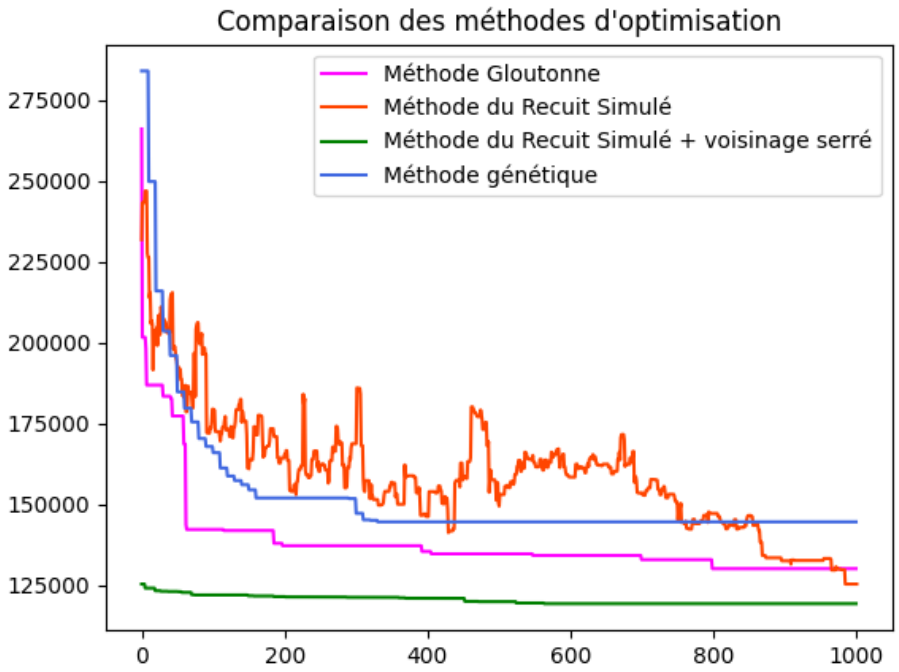
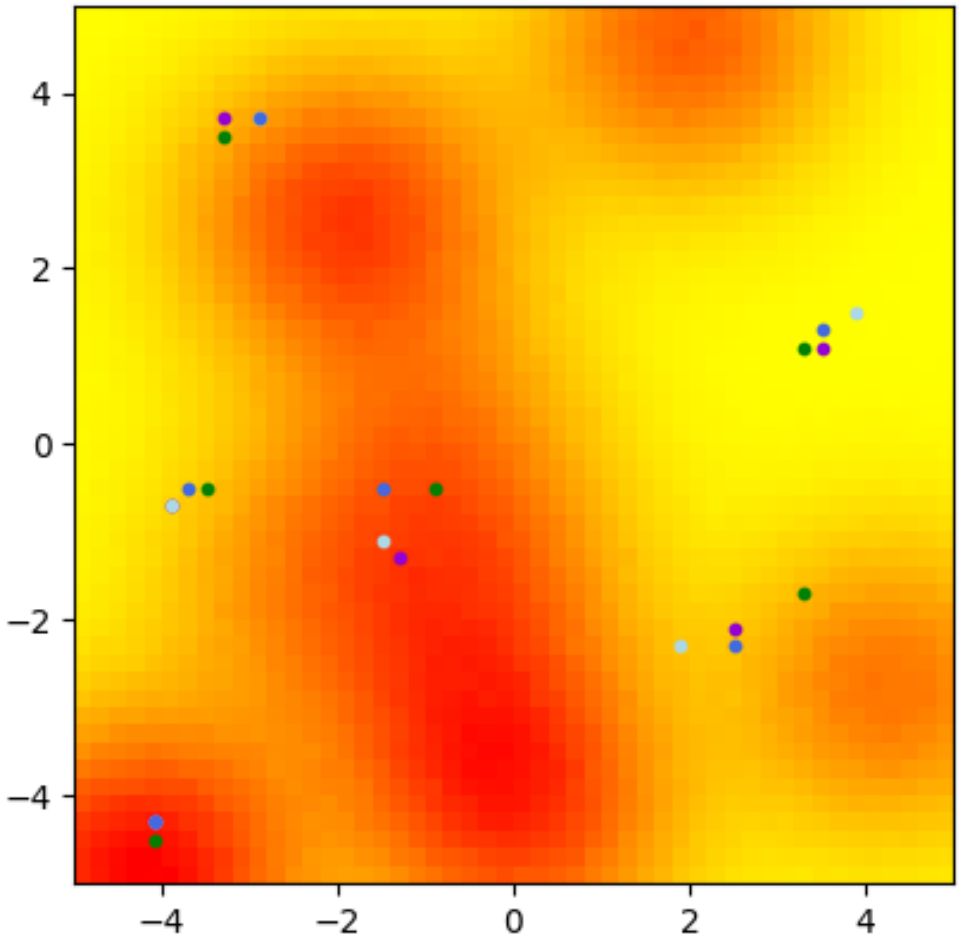


Figure: Exemple d'execution des deux algorithmes. L'abscisse est l'indice d'itération et l'ordonnée est la valeur de la fonction d'évaluation prise en la solution courante.

Comparaisons des algorithmes



Avantages et inconvénients de l'algorithme génétique

- Beaucoup de paramètres à ajuster correctement
- Algorithme très inconsistant : parfois d'excellents résultats, souvent pire que l'algorithme du gradient.
- Stagne relativement vite, et ne sort quasiment jamais de minimums locaux
- Extrèmement long comparé aux autres, a cause de la taille de la population
- La représentation d'un gène est très subjective : faut-il découper à la valeur ? à l'octet ? au bit ?

Avantages et inconvénients de l'algorithme génétique

- Algorithme très inconsistant : parfois d'excellents résultats

Application sur un exemple concret : La Sarthe

Région particulièrement touchée par les déserts médicaux :

- 13% de la population du département se trouve sans médecin traitant
- 90% des médecins généralistes et dentistes ne prennent plus de nouveaux patients

Le Pays de la Loire met publiquement et gratuitement à disposition des jeux des données sur son département

Application sur un exemple concret : La Sarthe

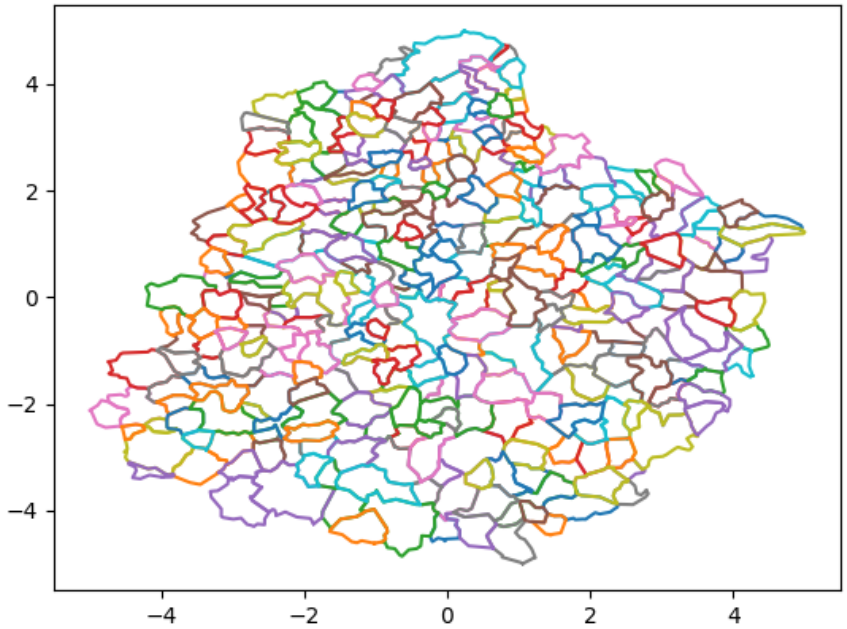


Figure: Communes de la Sarthe, représentées par des polygones

Source : https://data.paysdelaloire.fr/explore/dataset/227200029_les-communes-de-la-sarthe%40sarthe/table/?disjunctive.nom_comm

Application sur un exemple concret : La Sarthe

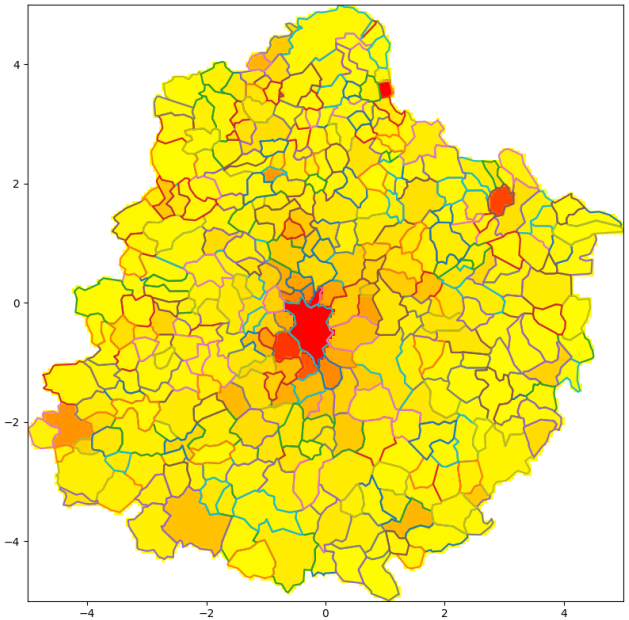


Figure: Carte de densité de la Sarthe, superposée aux contours des communes

Application sur un exemple concret : La Sarthe

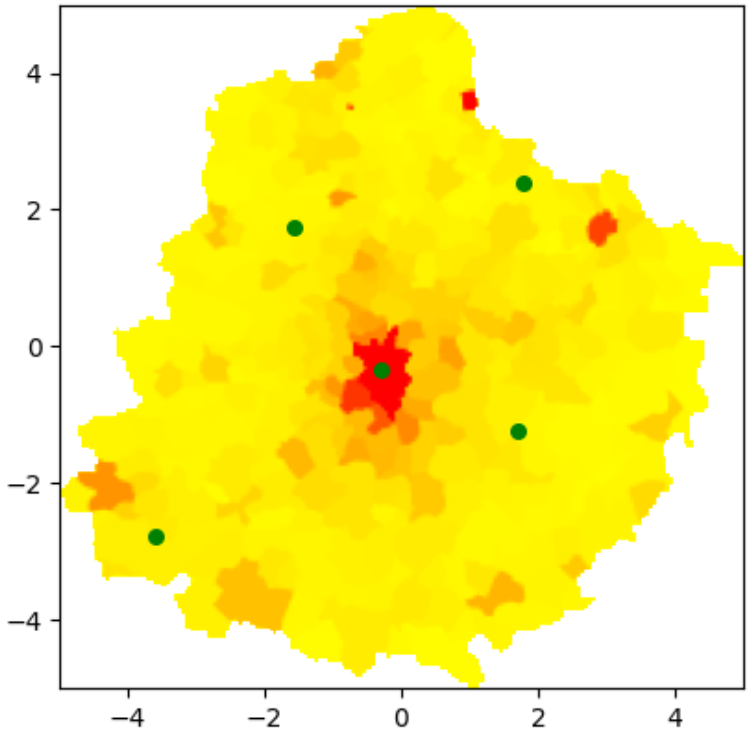


Figure: Meilleure solution trouvée par les différents algorithmes, pour 5 hôpitaux

Application sur un exemple concret : La Sarthe

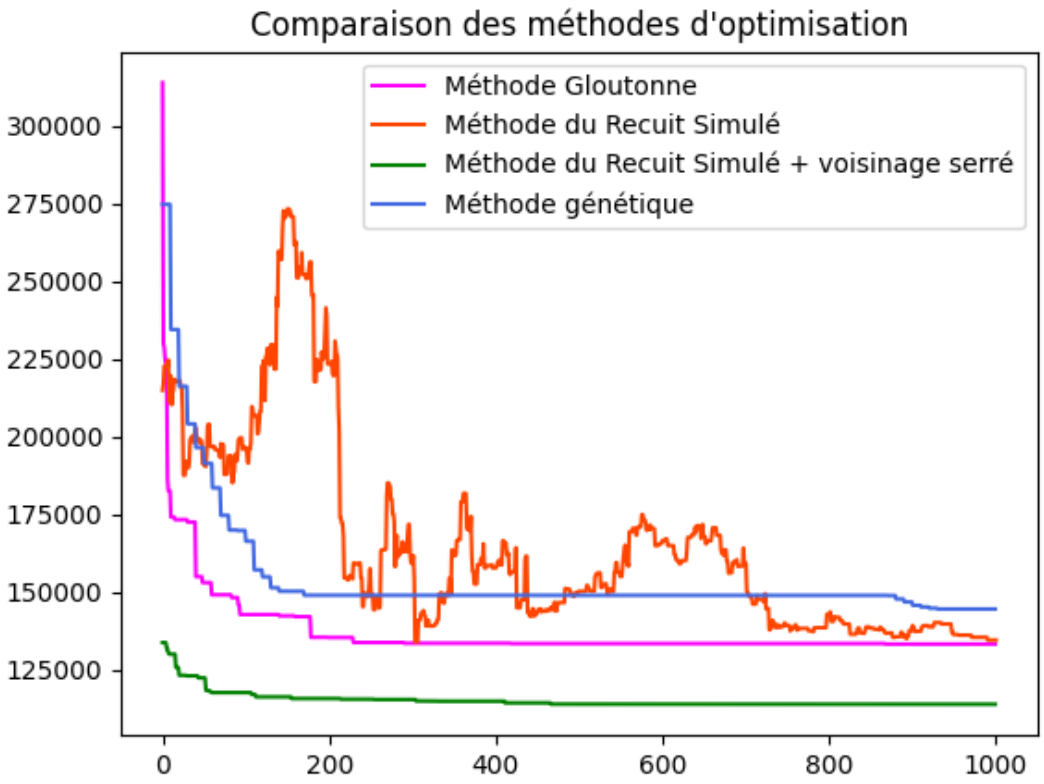


Figure: Comparaison des différents algorithmes pour la carte de densité de la Sarthe

Conclusion

Dans la majorité des cas :

- l'algorithme génétique était trop long pour des résultats insatisfaisant
- l'algorithme glouton était satisfaisant mais tombait trop souvent dans un minimum local
- l'algorithme du recuit simulé donnait des résultats souvent meilleurs que l'algorithme glouton
- l'algorithme du recuit simulé couplé à un algorithme glouton à voisinage serré donnait dans l'écrasante majorité des cas les meilleurs résultats

Conclusion

Merci de votre attention

Hypothèse de la distance

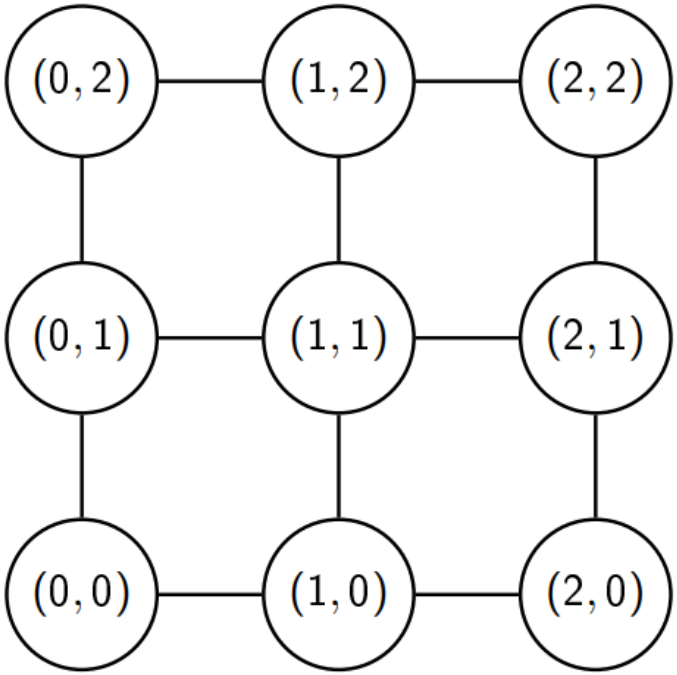


Figure: Modélisation de la carte de densité pour N = 3

- La distance entre deux sommets adjacents est la différence de leurs deux population
- On calcule la distance entre chaque sommet avec l'algorithme de Floyd-Warshall

Méthode "diviser pour régner"

A la manière d'un tri, on peut essayer d'appliquer une méthode dite "diviser pour régner" :

Définition 5.1

Une méthode dite "diviser pour régner" consiste en trois étapes

1. On divise le problème en plusieurs sous-problèmes similaires mais de taille inferieure.
2. On résout les problèmes plus petits par récurrence ou directement si leur taille est très petite.
3. On détermine une solution du problème en fonction des solutions des plus petits problèmes.

L'impossiblité d'une méthode "diviser pour régner"

On peut essayer d'appliquer ce raisonnement à notre problème :

1. On divise la matrice en 4 matrices carrées.
2. On résout par récurrence ces 4 sous-problèmes.
3. On détermine une solution du problème en fonction des solutions des plus petits problèmes.

L'impossiblité d'une méthode "diviser pour régner"

Cependant le 3^{eme} point pose problème : comment déterminer une solution à partir des sous-solutions ?

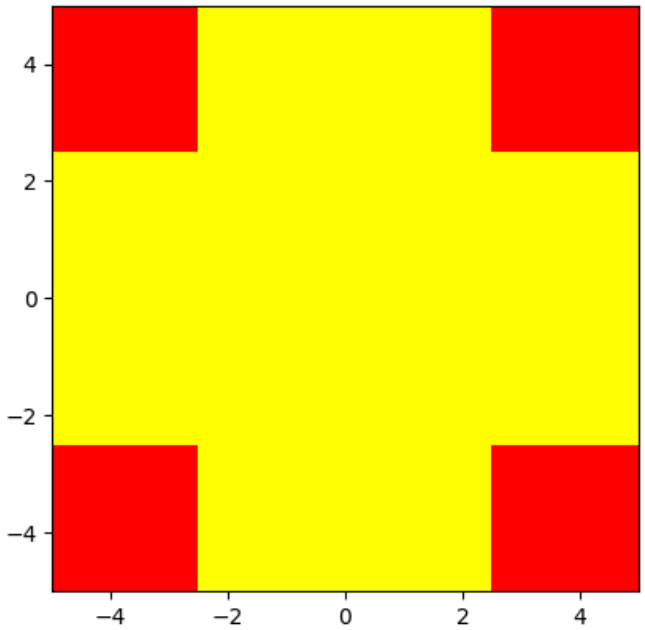


Figure: Exemple de carte de densité pour N = 4

L'impossiblité d'une méthode "diviser pour régner"

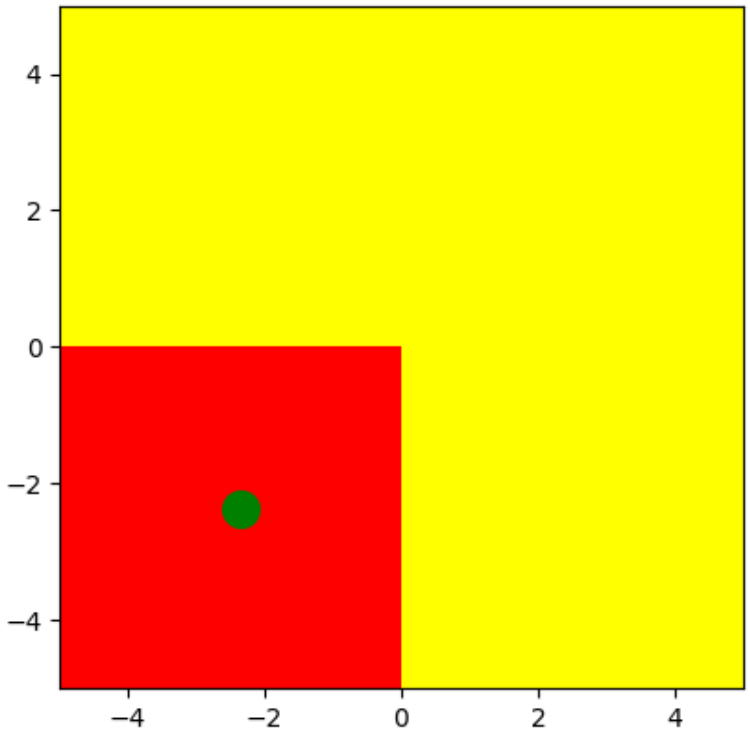


Figure: Exemple de sous-problème de carte de densité

L'impossiblité d'une méthode "diviser pour régner"

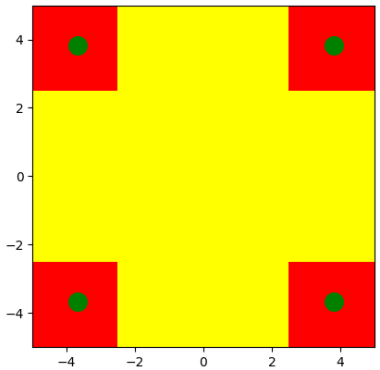


Figure: Superposition des sous-problèmes de carte de densité

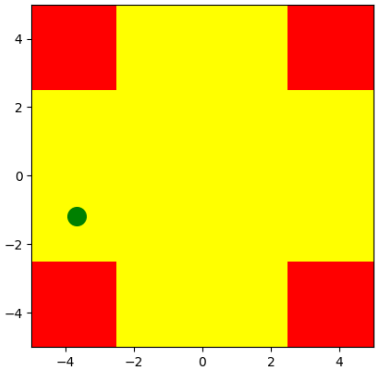


Figure: Solution exacte du problème de carte de densité

L'impossiblité d'une méthode "diviser pour régner"

- La solution exacte n'est pas une des solutions des sous-problèmes.
- La solution exacte n'est pas la moyenne ou la somme des solutions des sous-problèmes.

Conclusion

Il est impossible, ou du moins extrèmement difficile, de trouver la solution exacte au problème par une méthode "diviser pour régner", car le recollement des solutions est impossible.

Analogie avec la thermostatistique

Soit un métal quelconque :

- Lorsque l'on chauffe ce métal, les atomes commencent à se détacher de leurs liaisons et à se déplacer.
- Lorsque l'on refroidit ce métal, les atomes commencent à former de nouvelles liaisons.

Si le refroidissement se fait très lentement, la structure cristaline du métal est plus régulière et présente moins d'irrégularités.

Algorithme tabou

Définition 3.7

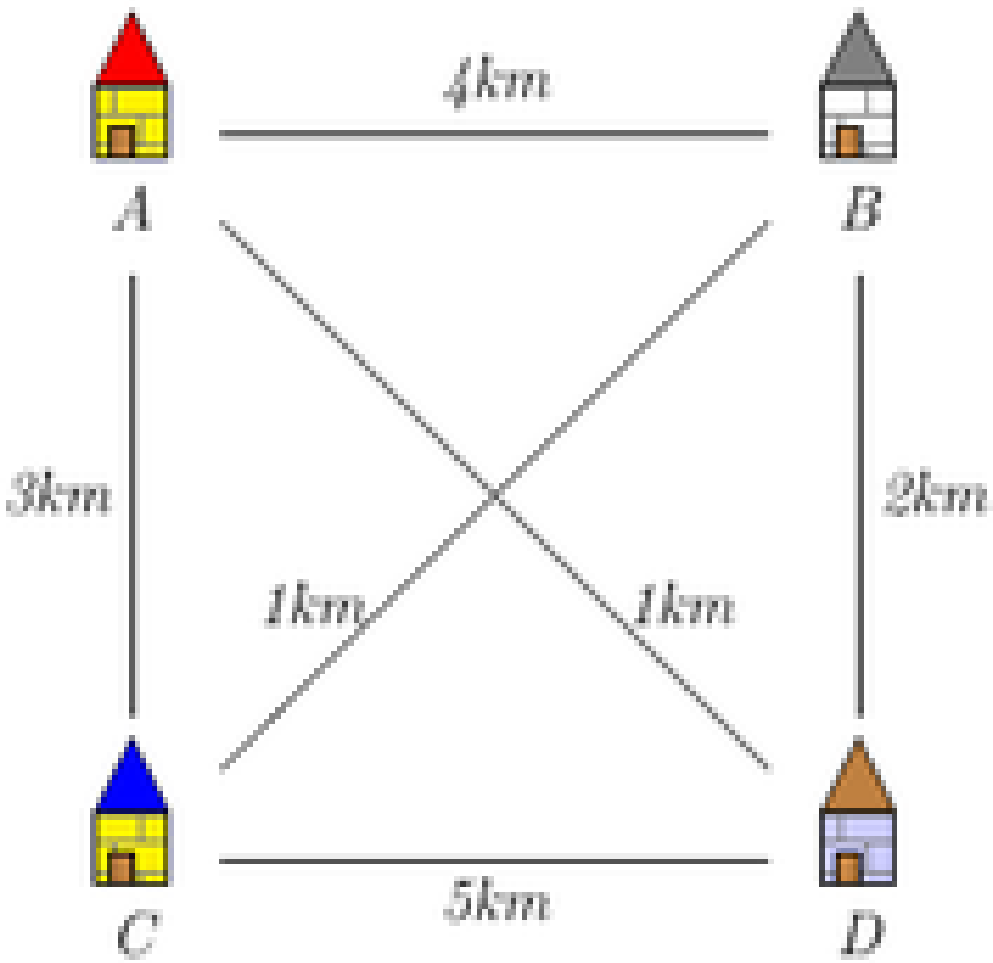
L'algorithme tabou est une méta-heuristique qui introduit une nouvelle variable : Une pile FIFO de taille fixe, et contenant des solutions interdites. Lorsque l'algorithme explore le voisinage, il n'a pas le droit d'avoir ces solutions interdites comme solution courante. A chaque nouvelle solution visitée, on ajoute la solution courante à la pile.

Algorithme tabou

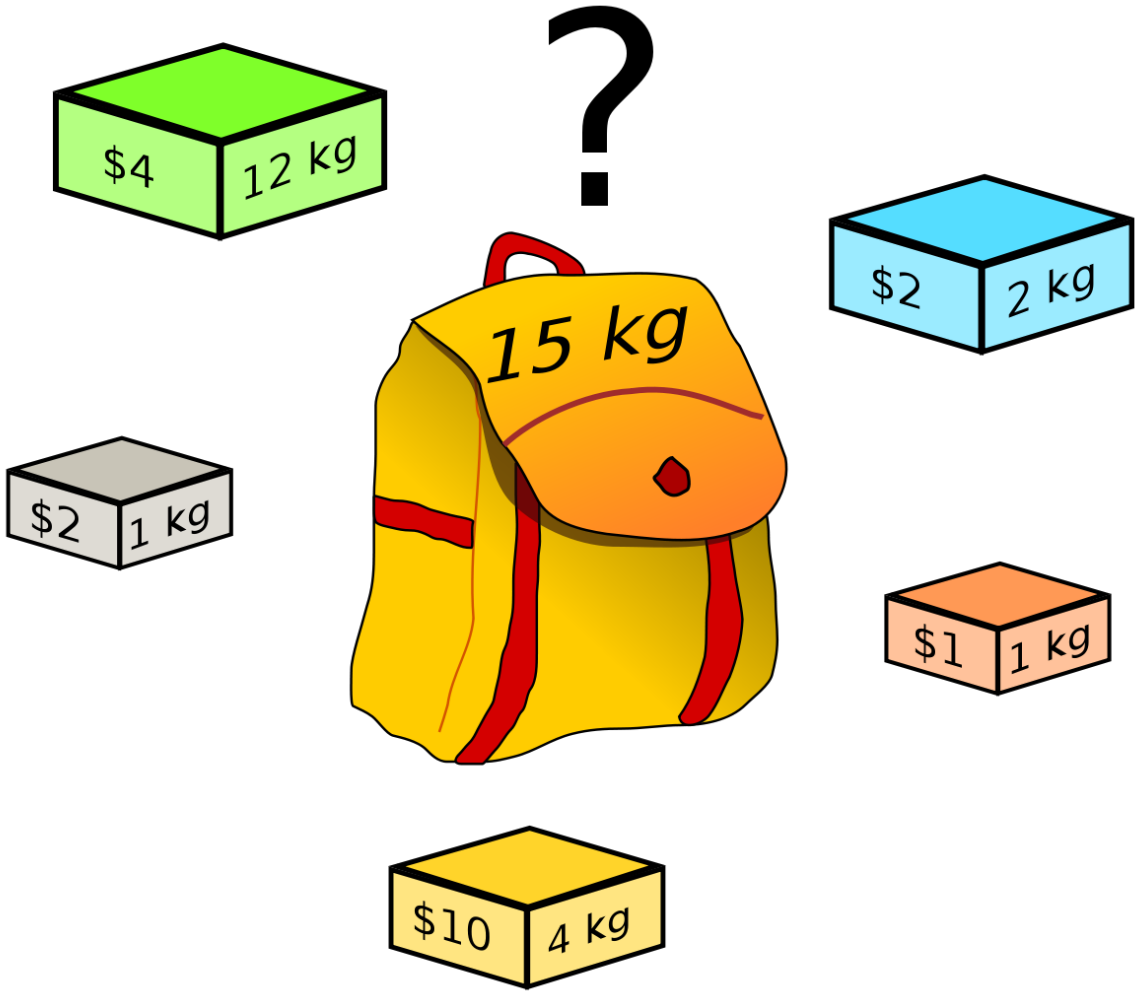
Causes possibles de l'échec :

- Une mauvaise taille de pile
- Une méthode non adaptée à ce type de problèmes
- Un voisinage trop grand
- Une erreur dans la transcription en python

Problème du voyageur de commerce



Problème du sac à dos



Code de la méthode gloutonne

```
solution = RandomSolution.GenerateRandomSolution(MatrixSize, NUM_OF_POINTS)

solutionScore = evaluate(Matrix, solution)[0]

#Pour chaque itération, on modifie une coordonnée d'un point, on regarde si le résultat est positif, et
si oui, on accepte la solution
for _ in range(NUM_OF_LOOPS):

    newSolution = GenerateNeighborSolution(solution, MatrixSize, NEIGHBOR_RANGE = NEIGHBOR_RANGE)

    newSolutionScore = evaluate(Matrix, newSolution)[0]

    #Si la solution est meilleure
    if newSolutionScore < solutionScore:
        solution = newSolution
        solutionScore = newSolutionScore

    scoreHistory.append(solutionScore)
```

Code de la méthode du recuit simulé

```
while currentTemp > ENDING_TEMP:

    newSolution = GenerateNeighborSolution(solution, MatrixSize, MatrixSize/4)
    newSolutionScore = evaluate(Matrix, newSolution)[0]

    scoreDifference = solutionScore - newSolutionScore

    if newSolutionScore < bestSolutionScore:
        bestSolution = newSolution
        bestSolutionScore = newSolutionScore

    temp = random.random()
    #Si le score est meilleur ou si le changement n'est pas trop brutal

    if scoreDifference > 0 OR temp < p0*np.exp((scoreDifference / currentTemp)):
        solution = newSolution
        solutionScore = newSolutionScore


    currentTemp -= GRADIENT

    scoreHistory.append(solutionScore)
```

Code de la méthode génétique

```
sizeOfCurrentPopulation = 0
for k in range(POPULATION_SIZE):
    if isinstance(population[k], Iterable):
        sizeOfCurrentPopulation = k + 1

#On reproduit la population actuelle
while sizeOfCurrentPopulation < POPULATION_SIZE:

    #On choisit deux parents
    parent1 = np.random.randint(0, sizeOfCurrentPopulation)
    parent2 = np.random.randint(0, sizeOfCurrentPopulation)
    while parent1 == parent2:
        parent2 = np.random.randint(0, sizeOfCurrentPopulation)

    separationIndex = np.random.randint(0, NUM_OF_POINTS - 1)

    #On crée deux enfants
    child1 = np.empty((NUM_OF_POINTS,), dtype=object)
    child2 = np.empty((NUM_OF_POINTS,), dtype=object)

    #On coupe des gènes des parents en deux, et on les injecte dans les enfants
    for k in range(NUM_OF_POINTS):
        if k <= separationIndex:
            child1[k] = population[parent1][k]
            child2[k] = population[parent2][k]
        else:
            child1[k] = population[parent2][k]
            child2[k] = population[parent1][k]

    #On ajotue les enfants à la population
    if sizeOfCurrentPopulation != POPULATION_SIZE - 1:
        population[sizeOfCurrentPopulation] = child1
        sizeOfCurrentPopulation += 1

    population[sizeOfCurrentPopulation] = child2
    sizeOfCurrentPopulation += 1
```

Code de la méthode génétique

```
for solution in population:

    #On mute quelques éléments de la population
    if random.random() < MUTATION_RATE:
        randomPoint = np.random.randint(0, NUM_OF_POINTS)
        solution[randomPoint] = RandomSolution.GenerateRandomPoint(MatrixSize)

#On filtre les pires candidats
scores = []
for solution in population:
scores.append(evaluate(Matrix, solution)[0])

#On supprime la moitié de la population
sortedScores = scores.copy()
sortedScores.sort()
thresholdScore = sortedScores[POPULATION_SIZE//2]

newPopulation = np.empty((POPULATION_SIZE,), dtype=object)
sizeOfCurrentPopulation = 0

for k in range(POPULATION_SIZE):
if scores[k] <= thresholdScore:
    newPopulation[sizeOfCurrentPopulation] = population[k]
    sizeOfCurrentPopulation += 1

#On définit la nouvelle population
np.copyto(population, newPopulation)
```

Code de la fonction d'évaluation

```
shape = matrix.shape
size = shape[0]
distances = np.empty((size**2,), dtype="int64") #array of 1 dimension
hopitaux = np.zeros(len(solution))

for x in range(size):
    for y in range(size):

        lowestDistance = np.inf
        closestHospital = np.inf
        for index, hospital in enumerate(solution):
            a,b = hospital
            distance = (a - x)**2 + (b - y)**2
            #On n'applique pas la racine carrée car on souhaite simplement comparer les distances

            if distance < lowestDistance:
                closestHospital = index
                lowestDistance = distance

        #Scale to percentages of max length
        distances[x*size + y] = math.sqrt(lowestDistance) * matrix[x][y] * 100 / (math.sqrt(2) * (size))
        hopitaux[closestHospital] = hopitaux[closestHospital] + 1
```

Code de la fonction d'évaluation

```
mean = distances.mean()

#Récupère le 5ème meilleur percentile
bestPercentile = np.percentile(distances, 5, axis=0, interpolation="higher")
median = np.percentile(distances, 50, axis=0, interpolation="higher")
worst = np.percentile(distances, 100, axis=0, interpolation="higher")
worstPercentile = np.percentile(distances, 95, axis=0, interpolation="higher")
saturation = max(hopitaux)

SCALAIRE_MEAN = 2
SCALAIRE_PERCENTILE5 = 1
SCALAIRE_PERCENTILE50 = 2
SCALAIRE_WORST = 1
SCALAIRE_PERCENTILE95 = 3
SCALAIRE_SATURATION = 10

#On applique les coefficients
mean *= SCALAIRE_MEAN
median *= SCALAIRE_PERCENTILE50
bestPercentile *= SCALAIRE_PERCENTILE5
worst *= SCALAIRE_WORST
worstPercentile *= SCALAIRE_PERCENTILE95
saturation *= SCALAIRE_SATURATION

score = mean + bestPercentile + median + worst + worstPercentile + saturation

return (score, [mean, bestPercentile, median, worst, worstPercentile, saturation])
```

Code de la modélisation de Sarthe

```
def GenerateSartheDensityMatrix(BINS = 50):

        polygons = []

        f = open('Sarthe.json')
        regions = json.load(f)

        for region in regions:
            sousregions = region["fields"]["geo_shape"]["coordinates"]

            population = region["fields"]["pop_m"]
            if len(sousregions) > 1: #Il y a plusieurs sous-régions
                sousPolygons = []
                for sousregion in sousregions:
                sousPolygons.append(GeneratePolygonFromRegionCoordinates(sousregion[0]))
                polygons.append((unary_union(sousPolygons), population))
            else:
                polygons.append((GeneratePolygonFromRegionCoordinates(sousregions[0]), population)
```

Code de la modélisation de Sarthe

```
increment = 10/BINS
for x in [-5 + k*increment for k in range(BINS)]:
    for y in [-5 + (10*k)/BINS for k in range(BINS)]:
        carre = Polygon([(x,y), (x + increment, y), (x + increment, y + increment), (x, y +
        increment)])

        population = 0
        for region in polygons:
            region_ratio = region[0].intersection(carre).area / region[0].area
            population += region_ratio*region[1]

        MAX_VALUE = 8000000/(BINS**2)
        population = (population if population < MAX_VALUE else MAX_VALUE)
        population = 1.01**population
        population = (0 if population == 1 else population)

        offset = (10/BINS)*(math.sqrt(2)/3)
        x_points.append(x + offset)
        y_points.append(y + offset)
        weights_points.append(population)
```

Bibliographie

- https://solidarites-sante.gouv.fr/systeme-de-sante-et-medico-social/masante2022/ (Définition d'une CPTS)
- https://www.ameli.fr/exercice-coordonne/exercice-professionnel/organisation-d-exercice-coordonne/constitution-d-une-cpts (Justification du role du placement d'une CPTS)
- https://tel.archives-ouvertes.fr/tel-00011623/document (justifier l'utilisation des MH)

Bibliographie

- https://homepages.laas.fr/huguet/drupal/sites/homepages.laas.fr.huguet/fi 2021-cours_Meta.pdf (Cours de Marie-Jo Huguet sur les MH, prof à l'INSA)
- Précis de recherche opérationnelle - 7ème édition (Décrit les différents types de MH)
- Métaheuristiques : Recuits simulé, recherche avec tabous, recherche à voisinages variables, méthodes grasp, algorithmes évolutionnaires, fourmis artificielles, essaims particulaires et autres méthodes d'optimisation.