Université Paris-Est Créteil Val-de-Marne
Faculté des sciences et technologies
M2 SDTDS 2020-2021

COMPTE RENDU

Optimisation

TP 01- Métaheuristiques

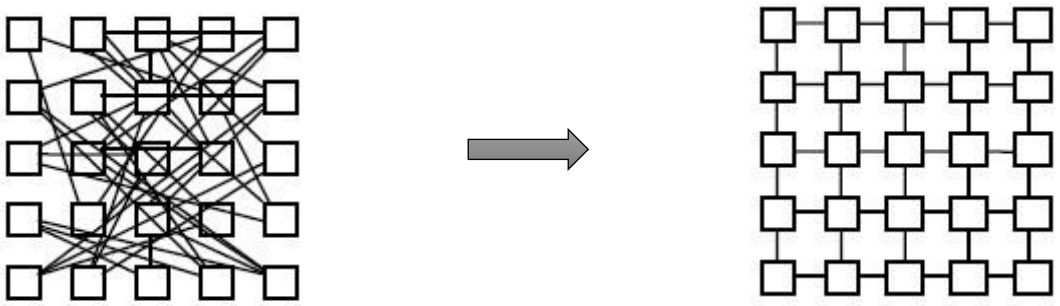
TP + compte rendu Réalisés par :

- **MERAD Fouad Fouzi**
- **BENSAMMAR Mohamed Aimene**
- **Attrassi Alaeddine**

Travaux Pratiques sur les « Métaheuristiques pour l'optimisation difficile »
Problème de placement de composants électroniques

Objectif : mettre au point un programme pour la résolution, à l'aide de l'algorithme du recuit simulé, d'un problème modèle de placement de 25 blocs de composants électroniques en des sites prédéterminés.

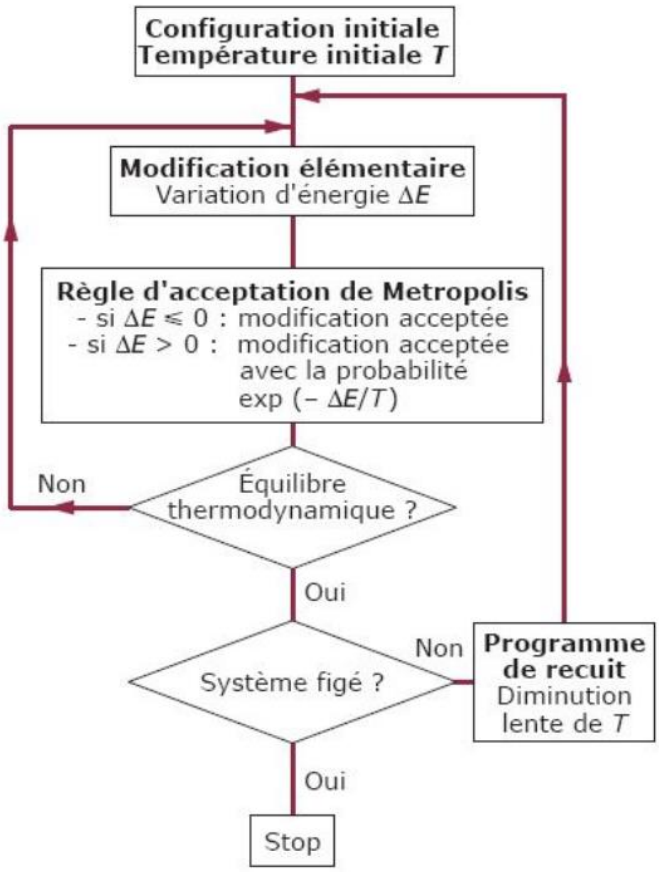
- Le seul mouvement autorisé est la permutation de deux blocs.
- La fonction objective est évaluée en sommant les longueurs en L des connexions.
- Le composant numéro i peut être placé sur n'importe quel nœud.
- Chaque composant est connecté à ses 4 « voisins » au maximum



1 : Configuration initiale

2 : Configuration finale (optimale)

Algorithme de recuit simulé :



Nous avons réalisé un programme avec le langage python sous l'environnement (IDE) Pycharm. Le code est divisé en 4 classes pour une meilleur organisation, lisibilité, utilisation et documentation.

Classe main.py :

- Fonction make_state :

  Génère aléatoirement une configuration initiale des composants

```python
# Générer un état initial de graphe aléatoirement
def make_state(g):
    state = mutation.randomize(mutation.gen(g, COLS, ROWS), NODES * NODES)
    return mutation.cost(g, state, EDGE_WEIGHT), state
```

- Fonction update :

  Afficher la configuration actuelle (console et interface graphique) de l'optimisation en cours à chaque intervalle des itérations effectuées + sauvegarder le graphique affiché en tant qu'image

```python
# Mise à jour de graphique affiché
def update(g, state, epoch, temp, acceptance=1, improvement=1):
    tmpsig = int(math.log10(TMAX) - math.log10(TMIN))
    print('Itération ' + ('%' + str(math.log10(EPOCH) + 1) + 'd') % epoch + ': Longueur optimale =', state[0], 'T◦ =', \
          (('{0:.' + str(tmpsig) + 'g}').format(temp) + ' ' * tmpsig)[:tmpsig + 1] + '\t', \
          '↓', '{0:.3g}'.format(acceptance * 100.), '↑', '{0:.3g}'.format(improvement * 100.))
    graph.show(g, state[1])
    plt.savefig("Itération-" + str(epoch) + ".png")
```

- Fonction move :

  La permutation des composants : permuter deux composants à la fois

  Calculer de nouveau le cout (langueur total obtenue) après la permutation

```python
# Permuter 2 composants (nœuds) et retourner l'état de graphe + le cout
def move(g, state):
    indiv = mutation.randomize(state[1])
    return mutation.cost(g, indiv, EDGE_WEIGHT), indiv
```

- Fonction optimize :

  Application de l'algorithme de recuit simulé afin d'atteindre la configuration optimale : c'est la fonction la plus compliquée et la plus longue elle réalise en étapes :

  Configuration initiale aléatoire des composants : Initialement, tous les composants doivent être dans le désordre afin de pouvoir appliquer l'algorithme du recuit simulé pour retrouver la configuration optimale ayant une longueur totale de 200.

  Permutation des composants : permuter deux composants à la fois.

  La distance Manhattan : obtenir la distance entre deux composant.

Température Initiale : définit au début avant l'exécution de l'algorithme par 25000, après on la déminue selon la fonction suivante :

Température = TMAX * math.exp (-math.log(TMAX / TMIN) * epoch / EPOCH)

TMAX : température maximale (25000)

TMIN : température minimale (0.1)

Epoch : itération en cours

EPOCH : Nombre Max des itérations à effectuer (10000)

Transformation élémentaire : permuter les composants aléatoirement dans la matrice et calculer la variation d'énergie pour tester sous différentes conditions ainsi arriver à trouver un équilibre thermodynamique.

Système figé : Une fois l'équilibre thermodynamique est atteint, on test avec des valeurs de température différentes et avec 0 acceptions pour arrêter l'algorithme et enfin visualiser le graphe et voir l'ordre des composants avec la longueur totale (cout).

Si la configuration optimale n'est pas atteinte on reboucle en diminuant la valeur de la température à chaque itération.

Solution optimale : Une fois que l'algorithme s'arrête il nous affiche automatiquement le résultat des longueurs total dans le graph et enregistre une capture sous un fichier image résultat.png.

```
# Appliquer l'algorithme de récuit simulé sur le graphe
def optimize(g):
    epoch = 0
    state = make_state(g)
    prev, best = copy.deepcopy(state), copy.deepcopy(state)

    tfactor = -math.log(TMAX / TMIN)
    temp = TMAX

    trials, accepts, improves = 0, 0, 0
```

```
    if UPDATES > 0:
        freq = EPOCH / UPDATES
        update(g, state, epoch, temp)

    while epoch < EPOCH:
        temp = TMAX * math.exp(tfactor * epoch / EPOCH)
        state = move(g, state)

        costdiff = state[0] - prev[0]
        trials += 1
        if costdiff > 0 and math.exp(-costdiff / temp) < random.random():
            state = copy.deepcopy(prev)
        else:
            accepts += 1
            if costdiff < 0:
                improves += 1
            prev = copy.deepcopy(state)
            if state[0] < best[0]:
                best = copy.deepcopy(state)

        epoch += 1

        if UPDATES > 1 and epoch // freq > (epoch - 1) // freq:
            update(g, state, epoch, temp, accepts / trials, improves / trials)
            trials, accepts, improves = 0, 0, 0

    return best
```

Classe random_.py :

- Fonction get :

```
# Retourne le nombre aléatoire à virgule flottante
def get():
    return random.Random(_seed)
```

- Fonction seed :

```
# Retourne la variable qui permet d'initialiser le générateur de nombres aléatoires
def seed():
    return _seed
```

Classe mutation_.py :

- Fonction gen :

```
# Retourner le numéro de ligne et de la colonne de chaque nœud
def gen(g, cols, rows):
    return [(i % cols, i // rows) for i in range(0, len(g.nodes()))]
```

- Fonction randomize :

Effectuer une permutation de deux composants :

```python
# Retourner une permutation des 2 emplacements obtenus aléatoirement (parmi les emplacements des nœuds)
def randomize(i, n=1):
    for j in range(0, n):
        first = _random.randrange(len(i))
        second = _random.randrange(len(i))
        while first == second:
            second = _random.randrange(len(i))
        i[first], i[second] = i[second], i[first]
    return i
```

- Fonction cost :

Retourner la longueur totale des connexions

```python
# Retourner le cout entre tous les nœuds (longueur des connexions)
def cost(g, indiv, weight):
    dist = 0
    for u, v in g.edges():
        dist += (abs(indiv[u][0] - indiv[v][0]) + abs(indiv[u][1] - indiv[v][1])) * weight
    return dist
```

Classe graph_.py :

- Fonction show :

Afficher l'interface graphique de schéma des composants

```python
# Affichage graphique de graphe (nœuds avec étiquettes, connexions)
def show(g, pos=None, rows=None, cols=None, orig=(0, 0)):
    if pos is None:
        pos = [((i % cols), i // rows) for i in range(0, cols * rows)]
    plt.clf()
    p = [(x + orig[0], y - orig[1]) for x, y in pos]
    nx.draw_networkx_nodes(g, p, node_color="#00b4d9")
    nx.draw_networkx_edges(g, p)
    nx.draw_networkx_labels(g, p)
    plt.axis('off')
    plt.pause(0.01)
```

- Fonction regular_grid :

Création des connexion entre les composants dans l'interface graphique

```python
# Ajouter les connexions entre les nœuds
def regular_grid(cols, rows):
    g = nx.Graph()
    g.add_nodes_from(list(range(0, rows * cols)))

    for i in range(0, cols - 1):
        for j in range(0, rows - 1):
            g.add_edge(i + j * cols, i + 1 + j * cols)
            g.add_edge(i + j * cols, i + (j + 1) * cols)

    for i in range(0, rows - 1):
        g.add_edge((cols - 1) + i * cols, (cols - 1) + (i + 1) * cols)

    for i in range(0, cols - 1):
        g.add_edge(i + (rows - 1) * cols, i + 1 + (rows - 1) * cols)
    return g
```

Résultat d'exécution de programme (expérimentation) :

Liste des captures obtenues lors de l'exécution de l'algorithme de recuit simulé (chaque 500 itérations) :

Itération 0 : étant la configuration initiale obtenue aléatoirement

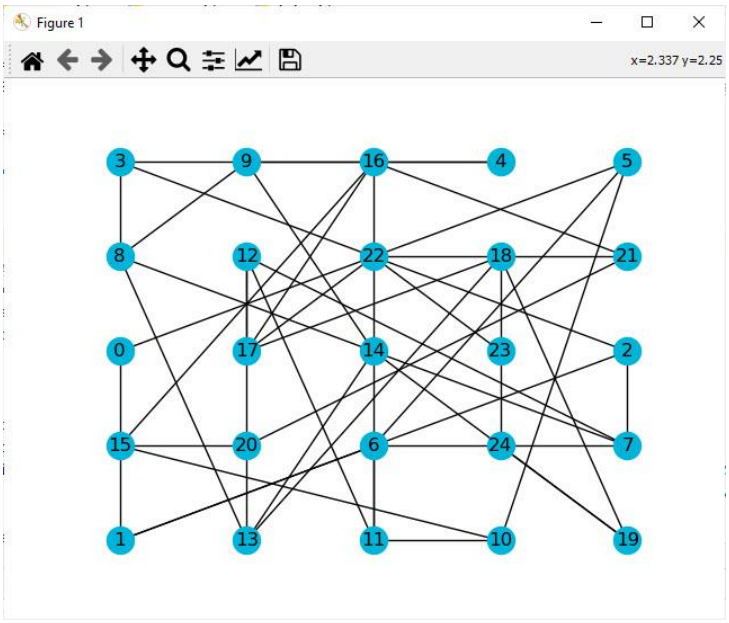
Itération (500 à 9500) : l'ensemble des configurations obtenues pendant l'exécution

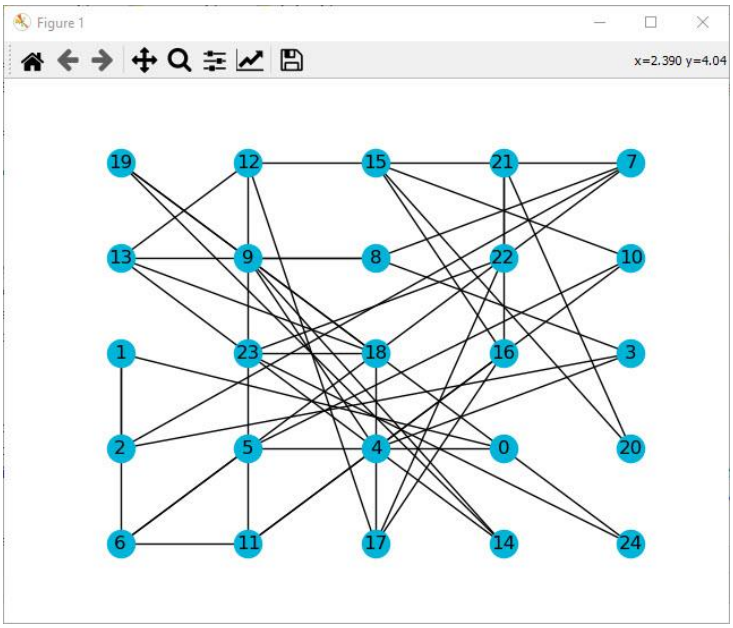
Itération 10000 : étant la configuration optimale

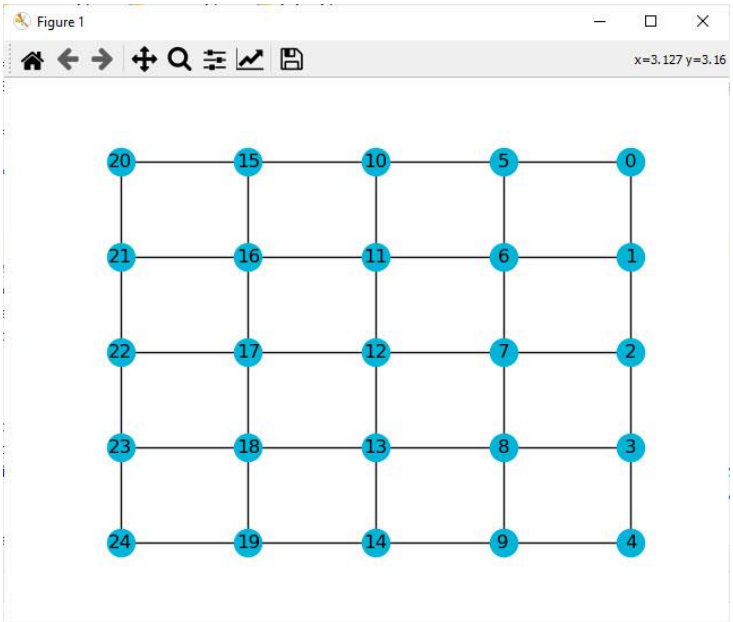
Affichage console :

```
Itération     0: Longueur optimale = 665 T◦ = 25000      √ 100 ↑ 100
Itération   500: Longueur optimale = 635 T◦ = 13446      √ 100 ↑ 43.8
Itération  1000: Longueur optimale = 750 T◦ = 7222.5     √ 100 ↑ 43.8
Itération  1500: Longueur optimale = 725 T◦ = 3879.6     √ 100 ↑ 43.6
Itération  2000: Longueur optimale = 665 T◦ = 2084       √ 99 ↑ 43.4
Itération  2500: Longueur optimale = 630 T◦ = 1119.4     √ 99.4 ↑ 44.6
Itération  3000: Longueur optimale = 710 T◦ = 601.31     √ 99.6 ↑ 44.8
Itération  3500: Longueur optimale = 640 T◦ = 323        √ 97 ↑ 40.2
Itération  4000: Longueur optimale = 610 T◦ = 173.5      √ 95 ↑ 40.2
Itération  4500: Longueur optimale = 580 T◦ = 93.198     √ 92 ↑ 37.2
Itération  5000: Longueur optimale = 725 T◦ = 50.062     √ 86.8 ↑ 34.4
Itération  5500: Longueur optimale = 600 T◦ = 26.891     √ 76.6 ↑ 31
Itération  6000: Longueur optimale = 480 T◦ = 14.445     √ 56.2 ↑ 21.6
Itération  6500: Longueur optimale = 405 T◦ = 7.7592     √ 27.8 ↑ 11.4
Itération  7000: Longueur optimale = 350 T◦ = 4.1679     √ 15.4 ↑ 5.8
Itération  7500: Longueur optimale = 285 T◦ = 2.2388     √ 4.8 ↑ 1.8
Itération  8000: Longueur optimale = 235 T◦ = 1.2026     √ 2.2 ↑ 1.2
Itération  8500: Longueur optimale = 215 T◦ = 0.646      √ 1.4 ↑ 0.4
Itération  9000: Longueur optimale = 200 T◦ = 0.347      √ 0.2 ↑ 0.2
Itération  9500: Longueur optimale = 200 T◦ = 0.1864     √ 0 ↑ 0
Itération 10000: Longueur optimale = 200 T◦ = 0.1001     √ 0 ↑ 0
Longueur optimale trouvée : 200
```

Affichage graphique :







Conclusion :

Le recuit simulé doit prendre beaucoup de paramètres pour pouvoir être satisfait du résultat attendu : le choix de la température initiale, le critère d'arrêt à déterminer, etc.

Au départ on doit choisir ces paramètres au hasard avant d'observer quelques résultats et de les ajuster.

Lorsque on lance plusieurs tests expérimentaux du recuit simulé, on peut déduire le résultat après s'être familiarisé sur ce programme.

L'avantage du recuit simulé est qu'il fournit de bon résultat pour un problème d'optimisation, par exemple dans le placement des composants électroniques dans des circuits.