Algorithmes de recherche locale

Résolution de Problèmes d'Optimisation
Master 1 informatique I2L / WeDSci

Sébastien Verel
verel@univ-littoral.fr
http://www-lisic.univ-littoral.fr/~verel

Université du Littoral Côte d'Opale
Laboratoire LISIC
Equipe OSMOSE

Plan

Problème d'optimisation

Problème d'optimisation

Un **problème d'optimisation** est un couple $(\mathcal{X}, f)$ avec :

- Espace de recherche : ensemble des solutions possibles,

$$\mathcal{X}$$

- fonction objectif : critère de qualité (ou de non-qualité)

$$f : \mathcal{X} \to \mathbb{R}$$

Résoudre un problème d'optimisation

Trouver la (ou les) meilleure solution selon le critère de qualité

$$x^\star = \text{argmax}_{\mathcal{X}}\ f$$

Problème d'optimisation

Problème d'optimisation

Un **problème d'optimisation** est un couple $(\mathcal{X}, f)$ avec :

- Espace de recherche : ensemble des solutions possibles,

$$\mathcal{X}$$

- fonction objectif : critère de qualité (ou de non-qualité)

$$f : \mathcal{X} \to \mathbb{R}$$

Résoudre un problème d'optimisation

Trouver la (ou les) meilleure solution selon le critère de qualité

$$x^\star = \text{argmax}_{\mathcal{X}}\ f$$

Mais, des fois, l'ensemble de toutes les meilleures solution, ou une bonne approximation, ou une solution "robuste", etc.

Contexte

Optimisation boite noire (Black box)

Nous ne pouvons connaitre que $\{(x_0, f(x_0)), (x_1, f(x_1)), ...\}$ donnés par un "oracle"
Aucune information sur la définition de la fonction objectif f n'est soit disponible ou soit nécessaire

$$x \longrightarrow$$  $$\longrightarrow f(x)$$

- Fonction objectif donnée par un calcul ou une simulation
- Fonction objectif peut être irrégulière, non différentielle, non continue, etc.

Typologie des problèmes d'optimisation

Classification

- **Optimisation combinatoire** : Espace de recherche dont les variables sont discrètes (cas NP-difficile)
- **Optimisation numérique (continue)** : Espace de recherche dont les variables sont continues
- **N'entrant pas dans les deux autres catégories** : combinaison discret/continue, programme, morphologie, topologie, etc.

Comment résoudre ce genre de problèmes ?

Search algorithms

Principle

Enumeration of the search space

A lot of ways to enumerate the search space :

- Exact methods :
  $A^\star$, Brand&Band, ...
- Approximation algorithms :
  construction of solution with performance guarante
- Using random sampling :
  Monte Carlo technics
- Heuristics and metaheuristics

Heuristiques

Définition : Heuristique

Algorithme de résolution dont

la conception repose sur l'expérience du concepteur.

Heuristiques

Définition : Heuristique

Algorithme de résolution dont
la conception repose sur l'expérience du concepteur.

Souvent :

- Pas de garantie d'obtenir une solution optimale

On désire toutefois :

- Le plus souvent possible une solution proche de l'optimalité
- Le moins souvent possible un mauvaise solution (différent !)
- Une complexité "raisonnable"
- De la simplicité d'implémentation :
  code simple dans les versions de base...

Metaheuristiques

H_θ

Peu probable qu'un algorithme puisse résoudre tout problème

Définition

Métaheuristique : Ensemble d'heuristiques

2 façons de définir un ensemble d'heuristiques :

- définir un ensemble d'heuristiques à l'aide de paramètres
- définir une méthode générique de conception d'heuristique

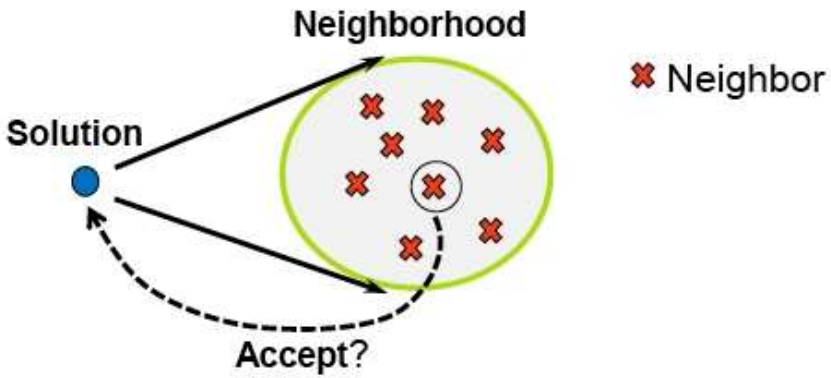
Principes et classification des métaheuristiques

Principes

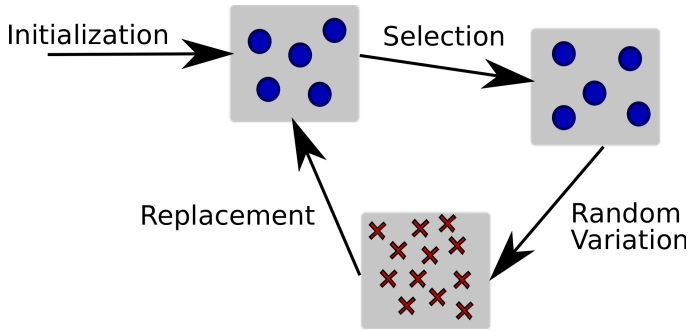
- Optimisation par essais/erreurs
- Itération du processus essais/erreurs

Classification :

Solution unique



Population de solution



Metaheuristisques (1)

Algorithmes à **population de solutions**

- Algorithmes Evolutionnaires (EA) :
  J. Holland 1975 et même avant
- Algorithmes d'essaims particulaires (PSO) :
  R. Ebenhart et J. Kennedy 1995.
- Algorithmes de fourmis (ACO) :
  Bonabeau 1999

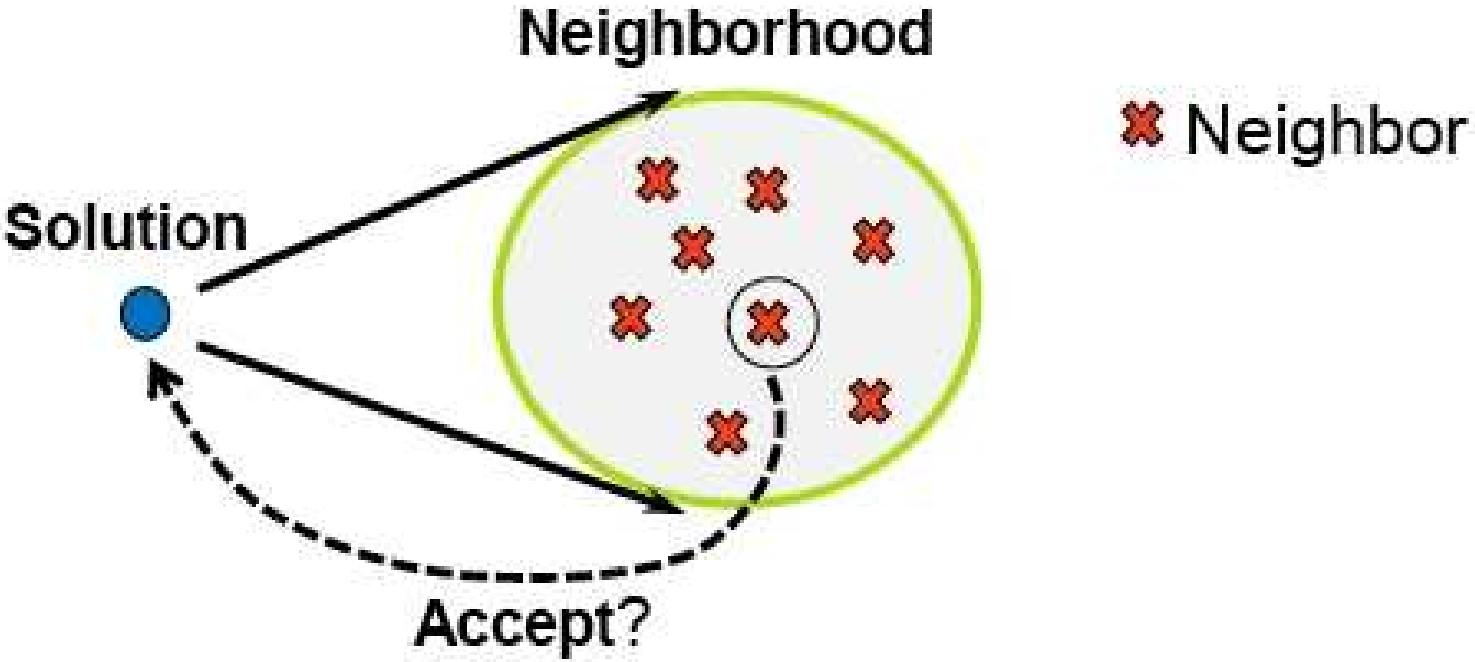
Metaheuristisques (2)

Algorithmes à **solution unique**

- Algorithmes de descente :
  Hill-Climber (HC) et variantes
- Recuit Simulé (SA) :
  Kirkpatrick *et al* 1983,
- Recherche Tabou (TS) :
  Glover 1986 - 89 - 90,
- Iterated Local Search
- ...

Metaheuristic with an unique solution : Local Search

- $\mathcal{X}$ set of solutions (search space)
- $f : \mathcal{X} \rightarrow \mathbb{R}$ objective function
- $\mathcal{V}(x)$ set of neighbor's solutions of x



Recherche Locale (LS) / Recherche locale stochastique

- $\mathcal{X}$ ensemble des solutions (espace de recherche),
- $f : \mathcal{X} \rightarrow \mathbb{R}$ fonction objectif à maximiser (ou coût à minimiser)
- $\mathcal{V}(x)$ ensemble des solutions voisines de x

Algorithme d'une Recherche Locale

Choisir solution initiale $x \in \mathcal{X}$
repeat
  choisir $x' \in \mathcal{V}(x)$
  if accept(x, x') **then**
    $x \leftarrow x'$
  end if
until critère d'arrêt verifié

Un exemple : Sac à dos avec contrainte

Problème du sac à dos :

Pour n objets, profits $(p_i)_{i\in\{1\dots n\}}$, poids $(w_i)_{i\in\{1\dots n\}}$

Codage binaire : objet i dans le sac $x_i = 1$, object hors sac $x_i = 0$.

Maximiser

$$z(x) = \sum_{i=1}^{n} p_i x_i$$

tel que :

$$w(x) = \sum_{i=1}^{n} w_i x_i \leqslant C$$

$$\forall i \in \{1 \dots n\}, x_i \in \{0,1\}^n$$

Sac à dos avec contrainte : fonction de pénalité

Fonction objectif avec pénalité

Soit le coefficient

$$\beta = \max\left\{\frac{p_i}{w_i} \ : \ i \in \{1 \dots n\} \text{ et } w_i \neq 0\right\} = \max_{i \in \{1 \dots n\} \text{ et } w_i \neq 0} \frac{p_i}{w_i}$$

Pour toute chaîne binaire de longueur n, $x \in \{0,1\}^n$

$$f(x) = \begin{cases} z(x) & \text{si } w(x) \leqslant C \\ z(x) - \beta \times (w(x) - C) & \text{si } w(x) > C \end{cases}$$

- Espace de recherche $\mathcal{X} =$

Sac à dos avec contrainte : fonction de pénalité

Fonction objectif avec pénalité

Soit le coefficient

$$\beta = \max\{\frac{p_i}{w_i} \ : \ i \in \{1 \dots n\} \text{ et } w_i \neq 0\} = \max_{i \in \{1 \dots n\} \text{ et } w_i \neq 0} \frac{p_i}{w_i}$$

Pour toute chaîne binaire de longueur n, $x \in \{0,1\}^n$

$$f(x) = \begin{cases} z(x) & \text{si } w(x) \leqslant C \\ z(x) - \beta \times (w(x) - C) & \text{si } w(x) > C \end{cases}$$

- Espace de recherche $\mathcal{X} = \{0,1\}^n$, Taille $\sharp\mathcal{X} =$

Sac à dos avec contrainte : fonction de pénalité

Fonction objectif avec pénalité

Soit le coefficient

$$\beta = \max\left\{\frac{p_i}{w_i} \ : \ i \in \{1 \dots n\} \text{ et } w_i \neq 0\right\} = \max_{i \in \{1 \dots n\} \text{ et } w_i \neq 0} \frac{p_i}{w_i}$$

Pour toute chaîne binaire de longueur n, $x \in \{0,1\}^n$

$$f(x) = \begin{cases} z(x) & \text{si } w(x) \leqslant C \\ z(x) - \beta \times (w(x) - C) & \text{si } w(x) > C \end{cases}$$

- Espace de recherche $\mathcal{X} = \{0,1\}^n$, Taille $\sharp\mathcal{X} = 2^n$

Sac à dos avec contrainte : fonction de pénalité

Fonction objectif avec pénalité

Soit le coefficient

$$\beta = \max\left\{\frac{p_i}{w_i} \ : \ i \in \{1 \dots n\} \text{ et } w_i \neq 0\right\} = \max_{i \in \{1 \dots n\} \text{ et } w_i \neq 0} \frac{p_i}{w_i}$$

Pour toute chaîne binaire de longueur n, $x \in \{0,1\}^n$

$$f(x) = \begin{cases} z(x) & \text{si } w(x) \leqslant C \\ z(x) - \beta \times (w(x) - C) & \text{si } w(x) > C \end{cases}$$

- Espace de recherche $\mathcal{X} = \{0,1\}^n$, Taille $\sharp\mathcal{X} = 2^n$

Résolution d'un problème d'optimisation combinatoire

Inputs

- Espace de recherche :

$$\mathcal{X} = \{0,1\}^n$$

- Function objectif :

$$f = \text{knapsack}$$

Goal

Find the best solution according to the criterium

$$x^\star = \text{argmax } f$$

Recherche Locale (LS) / Recherche locale stochastique

- $\mathcal{X}$ ensemble des solutions (espace de recherche),
- $f : \mathcal{X} \rightarrow \mathbb{R}$ fonction objectif à maximiser (ou coût à minimiser)
- $\mathcal{V}(x)$ ensemble des solutions voisines de x

Algorithme d'une Recherche Locale

Choisir solution initiale $x \in \mathcal{X}$
repeat
  choisir $x' \in \mathcal{V}(x)$
  if accept(x, x') **then**
    $x \leftarrow x'$
  end if
until critère d'arrêt verifié

Recherche aléatoire sur l'espace de recherche

Algorithme de Recherche Aléatoire

Choisir solution initiale $x \in \mathcal{X}$ aléatoirement uniformément sur $\mathcal{X}$.

repeat

choisir aléatoirement $x^{'} \in \mathcal{X}$

$x \leftarrow x^{'}$

until critère d'arrêt non verifié

Question

Est-ce qu'une recherche aléatoire est une recherche locale ?

Recherche aléatoire sur l'espace de recherche

Algorithme de Recherche Aléatoire

Choisir solution initiale $x \in \mathcal{X}$ aléatoirement uniformément sur $\mathcal{X}$.

repeat

choisir aléatoirement $x' \in \mathcal{X}$

$x \leftarrow x'$

until critère d'arrêt non verifié

- Voisinage est tout l'espace de recherche :
  $\mathcal{V}(x) = \mathcal{X}$
- Le choix de la solution voisine est aléatoire (loi uniforme)
- Les solutions sont toujours acceptées :
  $\text{accept}(x, x') = \mathit{true}$
- 1 ou plusieurs itérations :
  1 itération = 1 évaluation

Recherche aléatoire sur l'espace de recherche

Algorithme de Recherche Aléatoire (sur l'espace de recherche)

Choisir solution initiale $x \in \mathcal{X}$ aléatoirement uniformément sur $\mathcal{X}$.
Evaluer x avec f
$x_{best} \leftarrow x^{'}$
repeat
  Choisir aléatoirement $x^{'} \in \mathcal{X}$
  Evaluer $x^{'}$ avec f
  if $x^{'}$ est meilleure que x_{best} **then**
    $x_{best} \leftarrow x^{'}$
  end if
  $x \leftarrow x^{'}$
until critère d'arrêt vérifié
return x_{best}

Remarque : Evidement à cause du caractère aléatoire de la méthode, d'une exécution à l'autre, la qualité de la solution finale n'est pas la même.

Recherche aléatoire sur l'espace de recherche

Exercice

Questions :

- Coder la recherche aléatoire
- Evaluer les performances (qualité de la solution finale) de la recherche aléatoire en fonction du nombre d'évaluation.

  Méthode : Pour un nombre d'évaluation fixé, exécuter plusieurs fois la recherche aléatoire (au moins 30 fois), puis calculer les statistiques de la qualité des solutions finales obtenues.
- Observer et décrire la distribution des valeur de performances.

Conseil : enregistrer dans un fichier de type "csv", puis utiliser un logiciel dédié au calcul statistique comme R : `https://www.r-project.org/` pour calculer vos statistiques et dessiner les graphiques (voir notes de cours sur R).

Recherche Locale Aléatoire (marche aléatoire)

Heuristique d'**exploration** maximale

Recherche locale aléatoire
Marche aléatoire

Choisir solution initiale $x \in \mathcal{X}$
Evaluer x avec f
repeat
  choisir $x' \in \mathcal{V}(x)$ aléatoirement
  Evaluer x' avec f
  $x \leftarrow x'$
until Nbr d'éval. $>$ maxNbEval

- Algorithme inutilisable en pratique
- Algorithme de comparaison
- Opérateur local de base de nombreuses métaheuristiques
- Permet d'explorer la "forme" du paysage induit par le problème.

Voisinage des chaines binaires

Distance de Hamming

Nombre de différence entre 2 chaînes.

Voisinage de $x \in \{0,1\}^n$

Voisinage des chaines binaires

Distance de Hamming

Nombre de différence entre 2 chaînes.

Voisinage de $x \in \{0,1\}^n$

$\mathcal{V}(x)$: ensemble des chaînes binaires à une distance 1 de x.
"On modifie 1 seul bit"

Voisinage des chaines binaires

Distance de Hamming

Nombre de différence entre 2 chaînes.

Voisinage de $x \in \{0,1\}^n$

$\mathcal{V}(x)$: ensemble des chaînes binaires à une distance 1 de x.
"On modifie 1 seul bit"

Pour $x = 01101$, $\mathcal{V}(x) = \{$

Voisinage des chaines binaires

Distance de Hamming

Nombre de différence entre 2 chaînes.

Voisinage de $x \in \{0,1\}^n$

$\mathcal{V}(x)$: ensemble des chaînes binaires à une distance 1 de x.
"On modifie 1 seul bit"

$$\text{Pour } x = 01101,\ \mathcal{V}(x) = \{\ \begin{matrix} 01100, \\ 01111, \\ 01001, \\ 00101, \\ 11101 \end{matrix}\ \}$$

Voisinage des chaines binaires

Distance de Hamming

Nombre de différence entre 2 chaînes.

Voisinage de $x \in \{0,1\}^n$

$\mathcal{V}(x)$: ensemble des chaînes binaires à une distance 1 de x.
"On modifie 1 seul bit"

$$\text{Pour } x = 01101,\ \mathcal{V}(x) = \{\ \begin{matrix} 01100, \\ 01111, \\ 01001, \\ 00101, \\ 11101 \end{matrix}\ \}$$

- Taille du voisinage d'une chaîne binaire de longueur :

Voisinage des chaines binaires

Distance de Hamming

Nombre de différence entre 2 chaînes.

Voisinage de $x \in \{0,1\}^n$

$\mathcal{V}(x)$: ensemble des chaînes binaires à une distance 1 de x.
"On modifie 1 seul bit"

$$\text{Pour } x = 01101,\ \mathcal{V}(x) = \{\ \begin{matrix} 01100, \\ 01111, \\ 01001, \\ 00101, \\ 11101 \end{matrix}\ \}$$

- Taille du voisinage d'une chaîne binaire de longueur : n

Marche aléatoire

Exercice

- Coder la recherche locale aléatoire (marche aléatoire)
- Observer la forme du paysage en affichant le graphique de la dynamique de recherche :
  qualité de la solution $f(x)$ en fonction du nombre d'évaluations
- Le paysage vous semble-t-il rugueux ou régulier ? Est-ce que cela dépend de l'instance du problème ? Est-ce que cela dépend de la fonction de pénalité choisie ?

Hill-Climber Best Improvement (ou steepest-descent)

Heuristique d'**exploitation** maximale.

Hill Climber (best-improvement)

Choisir solution initiale $x \in \mathcal{X}$
Evaluer x avec f
repeat
  Choisir $x' \in \mathcal{V}(x)$ telle que $f(x')$ est maximale
  if x' strictement meilleur que x **then**
    $x \leftarrow x'$
  end if
until x optimum local

- Algorithme de comparaison
- Opérateur local de base de métaheuristique

Hill-Climber First Improvement

Hill-climber First-improvement (maximisation)

Choisir solution initiale $x \in \mathcal{X}$
Evaluer x avec f
repeat
  Choisir $x^{'} \in \mathcal{V}(x)$ aléatoirement tel que $f(x^{'}) > f(x)$ (si possible)
  Evaluer $x^{'}$ avec f
  if $f(x) < f(x^{'})$ **then**
    $x \leftarrow x^{'}$
  end if
until x optimum local **ou** nbEval $>$ maxNbEval

Hill-Climber First Improvement

Hill-climber First-improvement (maximisation)

Choisir solution initiale $x \in \mathcal{X}$
Evaluer x avec f
nbEval $\leftarrow 1$
repeat
  Choisir $x^{'} \in \mathcal{V}(x)$ aléatoirement
  Evaluer $x^{'}$ avec f et incrémenter nbEval
  while $f(x) \geqslant f(x^{'})$ **et** des voisins sont non visités **et** nbEval $\leq$ maxNbEval **do**
    Choisir $x^{'} \in \mathcal{V}(x)$ aléatoirement (avec ou sans remise)
    Evaluer $x^{'}$ avec f et incrémenter nbEval
  end while
  if $f(x) < f(x^{'})$ **then**
    $s \leftarrow x^{'}$
  end if
until x optimum local **ou** nbEval $>$ maxNbEval

Quelle est l'avantage de cet algorithme par rapport au Hill-Climber Best-improvement ?

Hill-Climber First Improvement

Hill-climber First-improvement (maximisation)

Choisir solution initiale $x \in \mathcal{X}$
Evaluer x avec f
nbEval $\leftarrow 1$
repeat
  Choisir $x^{'} \in \mathcal{V}(x)$ aléatoirement
  Evaluer $x^{'}$ avec f et incrémenter nbEval
  while $f(x) \geqslant f(x^{'})$ **et** des voisins sont non visités **et** nbEval $\leq$ maxNbEval **do**
    Choisir $x^{'} \in \mathcal{V}(x)$ aléatoirement (avec ou sans remise)
    Evaluer $x^{'}$ avec f et incrémenter nbEval
  end while
  if $f(x) < f(x^{'})$ **then**
    $s \leftarrow x^{'}$
  end if
until x optimum local **ou** nbEval $>$ maxNbEval

Le critère d'arrêt sur le nombre d'évaluation peut être changer (temps, qualité de la solution, etc.)

Hill-Climber First Improvement

Hill-climber First-improvement (maximisation)

Choisir solution initiale $x \in \mathcal{X}$
Evaluer x avec f
nbEval $\leftarrow 1$
repeat
  Choisir $x^{'} \in \mathcal{V}(x)$ aléatoirement
  Evaluer $x^{'}$ avec f et incrémenter nbEval
  while $f(x) \geqslant f(x^{'})$ **et** des voisins sont non visités **et** nbEval $\leq$ maxNbEval **do**
    Choisir $x^{'} \in \mathcal{V}(x)$ aléatoirement (avec ou sans remise)
    Evaluer $x^{'}$ avec f et incrémenter nbEval
  end while
  if $f(x) < f(x^{'})$ **then**
    $s \leftarrow x^{'}$
  end if
until x optimum local **ou** nbEval $>$ maxNbEval

Les comparaisons strictes ou non entre $f(s^{'})$ et $f(s)$ peuvent avoir de grandes conséquences sur les performances de l'algorithme.

Travaux pratiques !

Exercice

- Coder les recherches Hill-Climber best-improvement et Hill-Climber first-improvement
- Evaluer et comparer les performances de ces recherches : voir la suite pour la méthodologie

Comparaison d'algorithmes stochastiques

Une règle d'or

Ne jamais rien déduire d'une seule exécution de l'algorithme

On ne détermine pas si un dé est truqué en le lançant qu'une seule fois...

- Réaliser un nombre suffisant d'exécutions indépendantes
  typiquement au moins 30 runs (grand nombre)
  mais un nombre plus réduit peut être utilisé sous condition
- Utiliser un **test statistique** pour comparer les moyennes (ou les médianes) :
  - **Paramétrique** lorsque les distributions sont gaussiennes (à vérifier) :
    Test t de student, ANOVA à 1 facteur
  - **Non paramétrique** lorsque aucune hypothèse sous jacente :
    Test de la somme des rangs de Wilcoxon, Test de Mann-Whitney, Test de Kolmogorov-Smirnov
- Interpréter correctement les p-value obtenues

Comparaison de metaheuristiques

Technique

2 points de vue sont possibles :

- Pour un nombre d'évaluation fixé,
  on compare la qualité des solutions obtenues
- Pour une qualité fixée,
  on compare le nombre d'évaluation nécessaire pour l'obtenir

Comparaison de metaheuristiques

Technique

2 points de vue sont possibles :

- Pour un nombre d'évaluation fixé,
  on compare la qualité des solutions obtenues
- Pour une qualité fixée,
  on compare le nombre d'évaluation nécessaire pour l'obtenir

Problèmes

- Lorsque l'algorithme s'arrête avant le nombre d'évaluation fixé
- Lorsque le niveau de qualité n'est pas atteint
- Comment fixer le nombre d'évaluation, la qualité à atteindre ?

Comparaison de metaheuristiques

Problèmes

- Lorsque l'algorithme s'arrête avant le nombre d'évaluation fixé :
  - Considérer les évaluations "manquantes" comme perdues
  - Modifier les algorithmes en version "restart"
- Lorsque le niveau de qualité n'est pas atteint :
  - Mesurer et comparer seulement le taux de succès $\hat{p}_s$
  - Mesurer l'Expected Running Time (ERT) :
    $$E_s[T] + \frac{1-\hat{p}_s}{\hat{p}_s} T_{\text{limit}}$$
- Comment fixer le nombre d'évaluation, la qualité à atteindre ?
  - Etudier en fonction du nombre d'évaluation / de la qualité

Bibliographie

Sur les fonctions de pénalité pour le problème knapsack :

- Michalewicz, Zbigniew and Arabas, Jaroslaw, Genetic algorithms for the 0/1 knapsack problem, Methodologies for Intelligent Systems, pp. 134-143, 1994.
- He, Jun, and A Zhou, Yuren, A Comparison of GAs Using Penalizing Infeasible Solutions and Repairing Infeasible Solutions on Average Capacity Knapsack, Advances in Computation and Intelligence, pp. 100-109, 2007.

Pour aller plus loin sur les Hill-climbers :

- M. Basseur, A. Goëffon, Climbing Combinatorial Fitness Landscapes, Applied Soft Computing 30 :688-704, 2015.
- A. Goëffon, Modèles d'abstraction pour la résolution de problèmes combinatoires, Thèse d'Habilitation à Diriger des Recherches, 2014.
- Gabriela Ochoa, Sébastien Verel, Marco Tomassini, First-improvement vs. Best-improvement Local Optima Networks of NK Landscapes, In 11th International Conference on Parallel Problem Solving From Nature, pp. 104 - 113, 2010.