Algorithmes de recherche locale

Recherche Opérationnelle et Optimisation
Master 1

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

Université du Littoral Côte d'Opale
Laboratoire LISIC
Equipe OSMOSE

Plan

Definition of an optimization problem

Optimization problem

- Search space : Set of all feasible solutions,

$$\mathcal{X}$$

- Objective function : Quality criterium

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

Solving an optimization problem

Find the best solution according to the criterium

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

Definition of an optimization problem

Optimization problem

- Search space : Set of all feasible solutions,

$$\mathcal{X}$$

- Objective function : Quality criterium

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

Solving an optimization problem

Find the best solution according to the criterium

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

But, sometime, the set of all best solutions, good approximation of the best solution, good 'robust' solution...

Contexte

Black box Scenario

We have only $\{(x_0, f(x_0)), (x_1, f(x_1)), ...\}$ given by an "oracle"
No information is either not available or needed on the definition of objective function

- Objective function given by a computation, or a simulation
- Objective function can be irregular, non differentiable, non continous, etc.

Typologie des problèmes

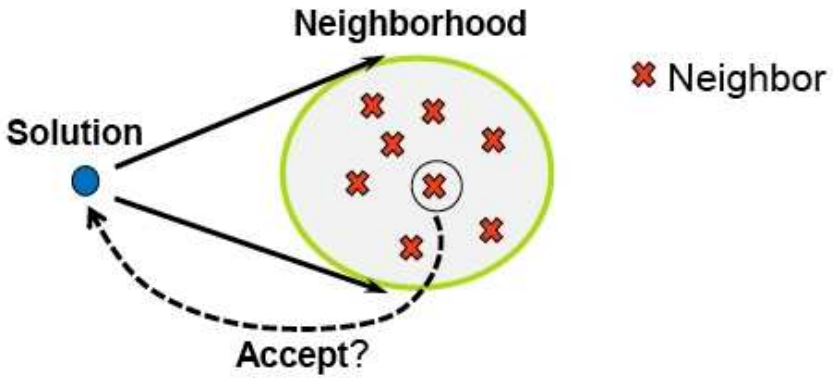
- **Optimisation combinatoire** : Espace de recherche dont les variables sont discrètes (cas NP-complet)
- **optimisation numérique (continue)** : Espace de recherche dont les variables sont continues

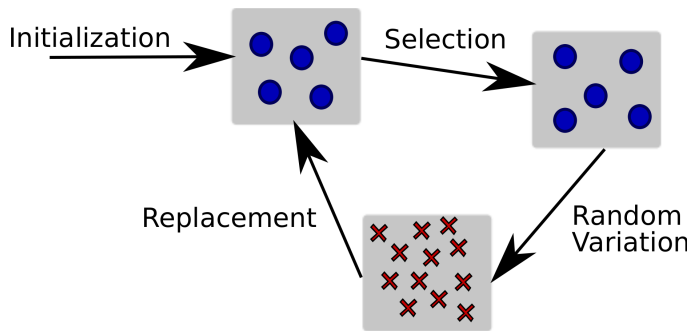
Search algorithms

Principle

Enumeration of the search space

- A lot of ways to enumerate the search space
- Using random sampling : Monte Carlo technics
- Local search technics :



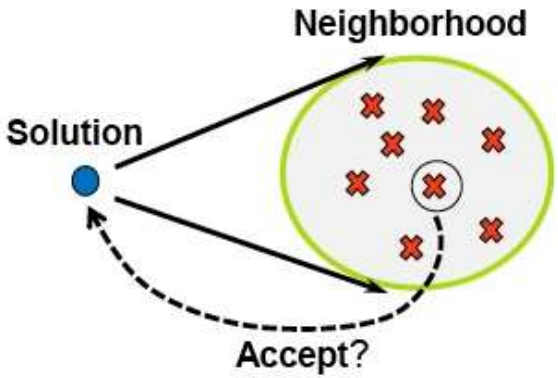


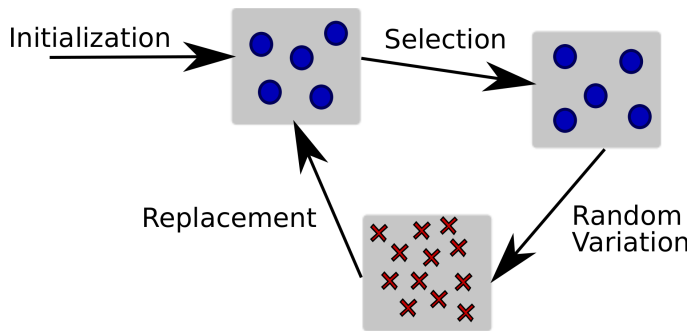
Search algorithms

Principle

Enumeration of the search space

- A lot of ways to enumerate the search space
- Using random sampling : Monte Carlo technics
- Local search technics :





- If objective function f has no propertie : random search
- If not...

Retour à MAX-SAT

Comment résoudre ce genre de problèmes ?

$$(x_1 \vee x_2) \wedge (\bar{x_2} \vee x_3) \wedge (x_1 \vee \bar{x_3})$$

$$(x_4 \vee x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee x_5) \wedge (\bar{x_2} \vee x_1 \vee x_5) \wedge (\bar{x_2} \vee x_5 \vee x_4) \wedge (x_1 \vee \bar{x_3} \vee \bar{x_4})$$

Retour à MAX-SAT

mathématiquement ou ...

- Exhaustivement $n = 20$, $n = 100$, ...
- Aléatoirement
- Construction de solution (plus tard)
- Méthodes exactes (plus tard)

ou ...

Heuristiques

Heuristique

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

Heuristiques

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 light en version de base...)

Metaheuristiques

Peu probable qu'un algorithme puisse résoudre tout problème

Métaheuristique

Ensemble d'heuristiques :

- regroupe des heuristiques dépendant de paramètres
- décrit une méthode de conception d'heuristique

de
un aveu d'impuissance
à
des techniques performantes d'optimisation difficile

Metaheuristisques de recherche locale

Algorithmes à **population de solutions**

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

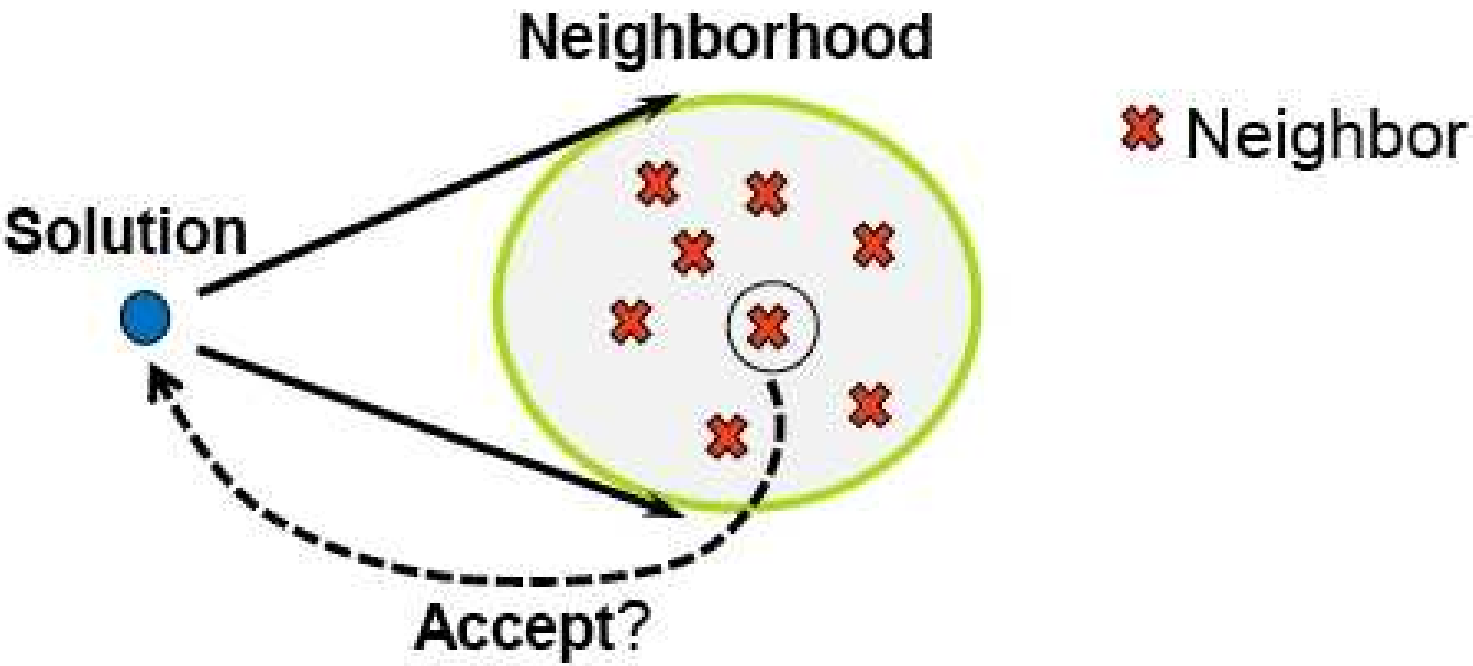
Metaheuristisques de recherche locale

Algorithmes à **solution unique**

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

Stochastic algorithms with unique solution (Local Search)

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



Recherche Locale (LS)

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

Algorithme d'une Recherche Locale

Choisir solution initiale $s \in \mathcal{S}$
repeat
  choisir $s' \in \mathcal{V}(s)$
  if accept(s, s') **then**
    $s \leftarrow s'$
  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 :

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

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

Sac à dos avec contrainte : fonction de pénalité

Fonction objectif avec pénalité

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

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

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

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

Sac à dos avec contrainte : fonction de pénalité

Fonction objectif avec pénalité

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

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

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

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

Sac à dos avec contrainte : fonction de pénalité

Fonction objectif avec pénalité

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

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

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

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

Sac à dos avec contrainte : fonction de pénalité

Fonction objectif avec pénalité

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

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

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

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

Exercice

- Coder, dans le langage que vous voulez, la fonction d'évaluation. L'instance d'un problème devra être définie à partir d'un fichier (nombre d'objects, profits, poids).
- Tester en affichant une solution et la valeur de la fonction

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

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

Recherche Locale (LS)

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

Algorithme d'une Recherche Locale

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

Recherche aléatoire sur l'espace de recherche

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

Choisir solution initiale $s \in \mathcal{S}$ aléatoirement uniformément sur $\mathcal{S}$.
repeat
  choisir aléatoirement $s' \in \mathcal{S}$
  $s \leftarrow s'$
until critère d'arrêt non verifié

- Voisinage $\mathcal{V} = \mathcal{S}$
- 1 seule itération, ou mieux aucune itération
- accept$(s, s') = \mathit{true}$

Recherche aléatoire sur l'espace de recherche

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

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

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.

Recherche Locale Aléatoire (marche aléatoire)

Heuristique d'**exploration** maximale

Recherche locale aléatoire
Marche aléatoire

Choisir solution initiale $s \in \mathcal{S}$
Evaluer s avec f
repeat
  choisir $s' \in \mathcal{V}(s)$ aléatoirement
  Evaluer s' avec f
  $s \leftarrow s'$
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"

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

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 *vs.* 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 $s \in \mathcal{S}$
Evaluer s avec f
repeat
  Choisir $s' \in \mathcal{V}(s)$ telle que $f(s')$ est maximale
  if s' strictement meilleur que s **then**
    $s \leftarrow s'$
  end if
until s optimum local

- Algorithme de comparaison
- Opérateur local de base de métaheuristique

Hill-Climber First Improvement

Hill-climber First-improvement

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

Hill-Climber First Improvement

Hill-climber First-improvement

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

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

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