Introduction aux méta-heuristiques, heuristiques hybrides et parallélisation MPI

Nicolas DUPIN

https://github.com/ndupin/ORteaching
http://nicolasdupin2000.wixsite.com/research

version du 20 mars 2022

Issu et étendu d'enseignements donnés à l'ENSTA Paris, Polytech' Paris-Saclay, et à l'université Paris-Saclay

Motivations (1)

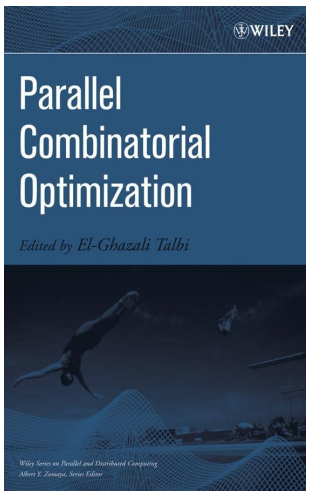
- Jusqu'ici, on a utilisé des méthodes d'optimisation exacte.
- But : trouver un algorithme pour converger le plus rapidement vers une solution prouvée optimale.
- Autre motivation : à une taille d'instances donnée, comment trouver les meilleures solutions en temps imparti (faible) ?
- Autrement dit : trouver les meilleures solutions possibles, sans preuve d'optimalité.
- Heuristiques/méta-heuristique : ensemble plus large de méthodes que les méthodes exactes.

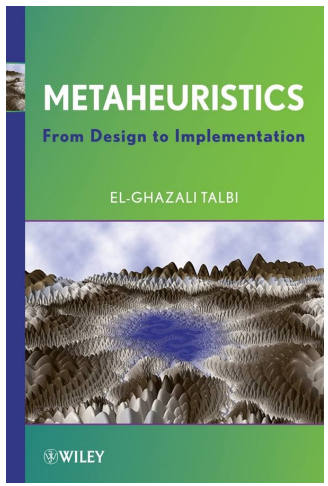
Motivations (2)

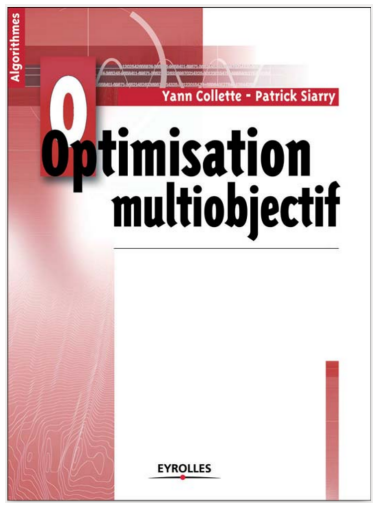
- Peut-on étendre les algorithmes de gradient à l'optimisation discrète ?
- Peut-on avoir des algorithmes légers pour résoudre des problèmes d'optimisation discrète dans des calculs embarqués (ex : robotique) ?
- Peut on utiliser des tels algorithmes dans un cadre multi-objectif ? (ça, c'est pour l'année prochaine en M2)
- Cadre d'application de la programmation parallèle, parallélisation à gros grains intéressante, cadre applicatif idéal pour de la parallélisation MPI.

Livres de référence (pour ce cours)

- Talbi, El-Ghazali. Metaheuristics : from design to implementation. Vol. 74. John Wiley & Sons, 2009.
- Des slides sont en ligne `http://www.lifl.fr/~talbi/metaheuristic/`
- Talbi, El-Ghazali, ed. Parallel combinatorial optimization. Vol. 58. John Wiley & Sons, 2006.
- Collette, Y., & Siarry, P. (2002). Optimisation multi-objectif. Editions Eyrolles.







Autres ouvrages de référence (en Anglais et en Français)

- Aarts & Lenstra, "Local Search for Combinatorial Optimization", Princeton University Press, 2003.
- Glover, F. W., & Kochenberger, G. A. (Eds.). (2006). Handbook of metaheuristics (Vol. 57). Springer Science & Business Media.
- Gendreau, M., & Potvin, J. Y. (Eds.). (2010). Handbook of metaheuristics (Vol. 2, p. 9). New York : Springer.
- Alba, E. Parallel metaheuristics : a new class of algorithms (Vol. 47). John Wiley & Sons. 2005
- Dréo, J., Pétrowski, A., Siarry, P., & Taillard, E. Métaheuristiques pour l'optimisation difficile. Editions Eyrolles, 2003.
- Siarry, P.. Métaheuristiques. Editions Eyrolles, 2014.
- in-Kao Hao, Christine Solnon. Méta-heuristiques et intelligence artificielle. Pierre Marquis, Odile Papini, Henri Prade. Algorithmes pour l'intelligence artificielle, Volume 2, Série Panorama de l'intelligence artificielle, Cépaduès, pp.1-19, 2014.`https://hal.archives-ouvertes.fr/hal-01313162/document`

Remarque, minimisation vs maximisation

- Dans les éléments généraux de ce cours, on considèrera sauf mention du contraire que l'on considère un problème de minimisation.
- Cela définira minimums globaux et minimums locaux.
- Aucune perte de généralité, en cas de problème de maximisation on se ramène à la minimisation en considérant min -f(x) au lieu de max f(x).
- Dans la plupart des cas, cela induira peu de changement de passer de min à max. Attention à l'algorithme de recuit simulé.

Plan

Plan

Définitions des heuristiques/méta-heuristiques

- Définition "heuristique" variable suivant champs d'applications : sociologie, philosophie, histoire, psychologie.
- En optimisation combinatoire, une heuristique est une méthode de calcul qui fournit rapidement une solution réalisable de bonne qualité mais pas nécessairement optimale à un problème d'optimisation combinatoire.
- Méta-heuristiques : niveau d'abstraction supérieur, méthodes permettant d'être adaptées à une large gamme de problèmes différents.
- Une heuristique est spécialisée à un problème spécifique.
- Une méta-heuristique est une famille générique d'heuristiques qu'il faut adapter à chaque problème.
- Pour illustrer ces concepts dans ce cours, problèmes du voyageur de commerce (TSP), coloration, cliques ou stables dans un graphe et p-median.

Principes généraux

- Les méta-heuristiques sont des stratégies qui permettent de guider la recherche de bonnes solutions.
- Les méta-heuristiques visent à explorer l'espace de recherche efficacement afin de déterminer des solutions (presque) optimales.
- Idée : Au lieu d'effectuer une recherche exhaustive parfois longue, on n'en explore qu'une partie, guidé par la fonction objectif.
- Points communs clés : encodage de l'espace des solutions, voisinages, intensification, diversification, mémoire.
- Les méta-heuristiques sont en général non-déterministes et ne donnent aucune garantie d'optimalité.

Pour illustrer ces concepts :

Exemples "simples" et classiques pour des méta-heuristiques :

- Problème du voyageur de commerce (TSP), et extension VRP.
- Clustering : cas p-median, vous connaissez l'algoritme de Lloyd pour k-means.

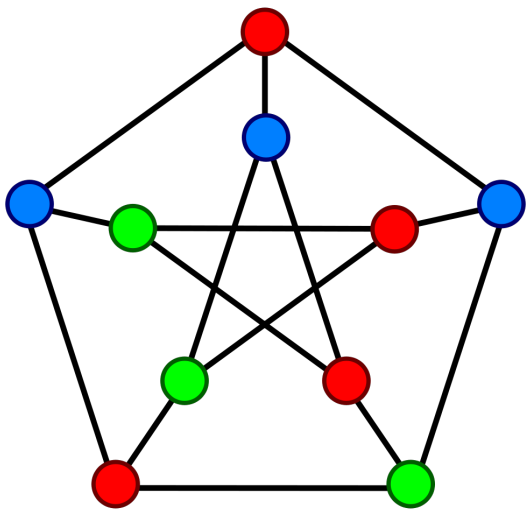
Exemples "plus difficiles" pour des méta-heuristiques :

- Cliques max/stable max dans un graphe (le même problème en passant au complémentaire)
- Problème de coloration des sommets

Exemple " encore plus difficile" pour des méta-heuristiques, car plus contraint :

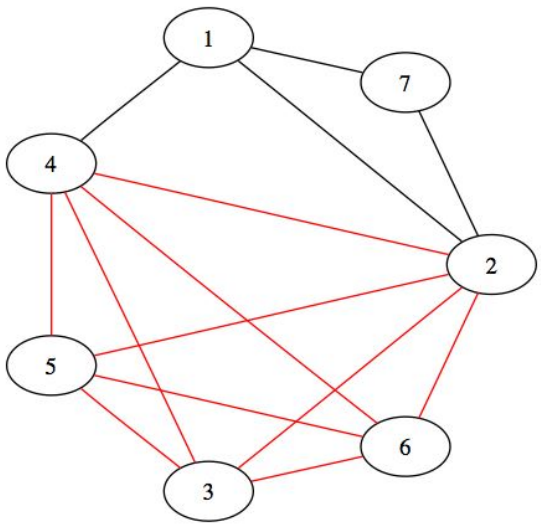
- TSPTW : TSP avec fenêtres de temps.

Rappels : problème de coloration de sommets d'un graphe



A partir d'un graphe donné, minimiser le nombre de couleurs à utiliser pour colorer tous les sommets du graphe, tq deux sommets voisins soient reliés par une couleur différente

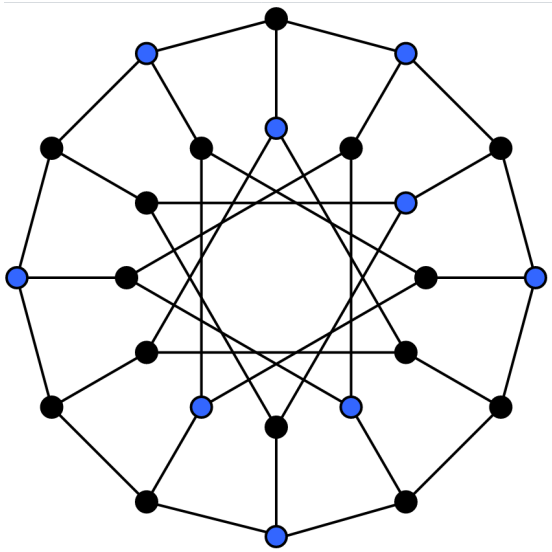
Recherche de clique de cardinal maximal d'un graphe



Problème de la clique MAXIMUM (Clique-MAX) :
A partir d'un graphe donné, trouver une clique de cardinal maximal, ie sélectionner un nomre maximal de sommets reliés entre eux deux à deux.

$\Longrightarrow$ C'est un minorant du nombre chromatique, optimum du problème de coloration de sommets

Recherche de stable de cardinal maximal d'un graphe



Problème du stable MAXIMUM (StableMAX) : A partir d'un graphe donné, trouver un stable de cardinal maximal, ie sélectionner un nombre maximal de sommets reliés entre eux deux à deux.

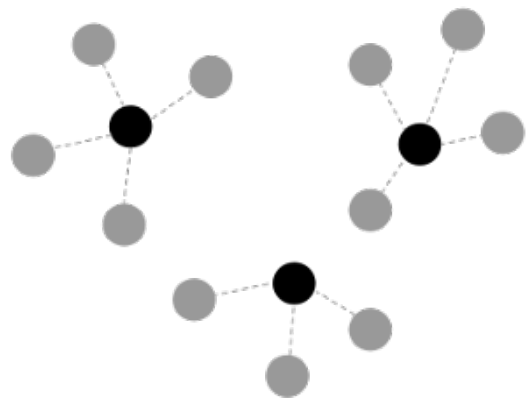
$\Longrightarrow$ Dans le problème de coloration de sommets, les couleurs définissent des stables.

$\Longrightarrow$ La recherche de stable (resp clique) de cardinal maximal revient à cherche une clique (resp stable) de cardinal maximal dans le graphe complémentaire

Rappel : p-median, k-medoids

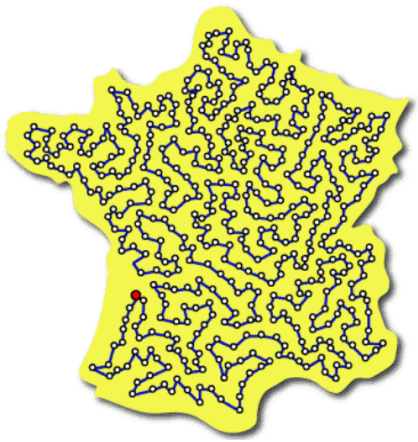
Données d'entrée : un ensemble X de N points dans $\mathbb{R}^d$, un entier $p < N$

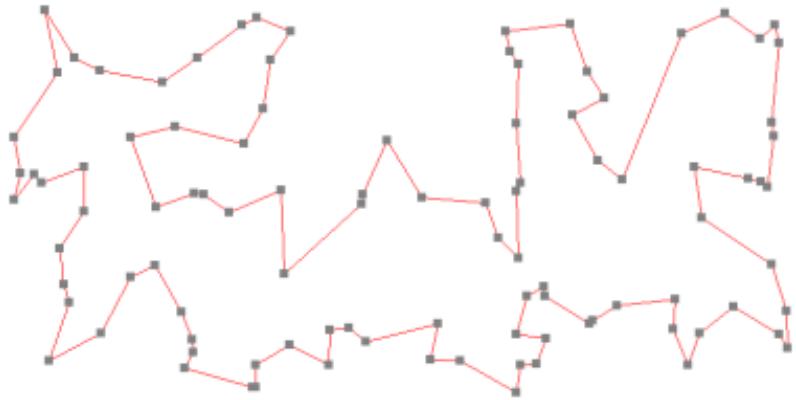
p-median : On veut former une partition de X en p sous-ensembles (= cluster), choisir p éléments représentants de chacun des clusters qui minimise la somme des distances de chacun des N points au représentant du cluster dont il appartient.



k-medoids : idem que k-median, sauf que l'on minimise la somme des carrés des distances. (Variante discrète de k-means, les k-moyennes)

Rappel problème du voyageur de commerce (TSP)





- Un voyageur de commerce doit transiter par plusieurs villes numérotées de 1 à N.
- Les distances entre deux villes $i, i' \in [\![1, N]\!]$ sont connues et notées $d_{i,i'}$.
- Minimiser la longueur d'un trajet reliant les N villes.

Contrainte additionnelle : TSPTW

- TSPTW : TSP avec contraintes de fenêtres de temps (TW, time windows)
- Contrainte additionnelle : On part de la ville 1 à une heure T_1.
- Pour chaque ville $i \in [\![2, N]\!]$, le temps d'arrivée à i doit être compris entre $\underline{T}_i$ et $\overline{T}_i$
- Les temps de parcours entre deux villes $i, i' \in [\![1, N]\!]$ sont connus et notés $t_{i,i'}$.
- Ex : tournée de livraison.
- Minimiser la longueur d'un trajet reliant les N villes en respectant les contraintes d'arrivées.

Encodage de solutions, règles générales

- Encodage : comment écrire l'espace des solutions ?
- En PLNE, on voyait déjà de nombreuses variantes apparaître.
- Sur les cas traités : encodage avec des binaires/booléens, bcp de variables.
- Pour les méta-heuristiques, on utilisera l'encodage le plus compact.
- Point clé : avoir un encodage le plus approprié aux contraintes.
- Encodage-décodage : certaines structures peuvent être redondantes dans l'encodage, ou recalculées rapidement à partir de certaines décisions, des variantes à considérer suivant les opérateurs heuristiques que l'on souhaite implémenter.

Encodage, exemple du p-median

- Encodage : comment décrire l'espace des solutions ?
- Un sous ensemble de points : les centres de clusters + les affectations points-clusters.
- ex : deux tableaux de taille N, un tableau de booléens pour désigner les centres et un tableau d'entiers à valeurs dans $[\![1; K]\!]$, les affectations.
- Le tableau de booléens a alors K valeurs 1 exactement.
- Variante : considérer un tableau de taille K, indiquant les centres. Attention à vérifier les non-répétitions avec un tel encodage.

Encodage-décodage, exemple du p-median

- Encodage : comment décrire l'espace des solutions ?
- Un sous ensemble de points : les centres de clusters + les affectations points-clusters.
- ex : deux tableaux de taille N, un tableau de booléens pour désigner les centres et un tableau d'entiers à valeurs dans $[\![1; K]\!]$, les affectations.
- Une fois les centres choisis, les points sont affectés aux centres les plus proches ! !

  Il suffit de stocker le tableau de booléens donnant les centres, les affectations optimales (et donc le coût associé associé au clustering) pour ces centres se calculent en $O(KN)$. (et l'implémentation du décodage est parallélisable)

Encodage, exemple TSP

- Encodage : comment décrire l'espace des solutions ?
- Mathématiquement : une permutation de $[\![1, N]\!]$, l'ordre compte
- On peut partir de 1, la solution est un cycle, la solution peut être encodée avec une permutation de $[\![2, N]\!]$
- Peut être encodé comme un vecteur, attention à ne pas introduire de bugs avec des répétitions !
- LocalSolver admet un conteneur spécifique pour les permutations, pour des heuristiques black-box basées sur ces structures standards.

Encodage-décodage, exemple TSPTW

- Variantes TSPTW, on aurait de plus en encodage les heures d'arrivées dans chaque ville : un tableau de taille $N - 1$
- Une fois un ordre de parcours défini, on peut calculer les horaires au plus tôt d'arrivée dans chaque ville, dans l'ordre de parcours, et en minimisant les temps d'attentes si le parcours fait arriver avant les dates au plus tôt $\underline{T}_i$.
- Le tableau des horaires n'est pas nécessaire pour exprimer la fonction objectif, minimiser la distance de parcours.
- Le tableau des horaires est nécessaire pour vérifier la contrainte d'horaires.
- Le calcul au plus tôt permet de valider les contraintes, ou de prouver que les contraintes horaires ne peuvent être toutes satisfaites.
- N.B : si on pénalisait les retards (pénalités financières à chaque retard par exemple), on considérerait les horaires dans l'encodage et pas un encodage-décodage.

Encodage, exemple stableMax,/coloration

- Encodage : comment décrire l'espace des solutions ?
- Une vecteur de $\{0, 1\}$ de taille $|V|$, indique si chacun des sommets est considéré dans le stable.
- Variante : Une liste d'élément sélectionnés.
- Dans les deux cas, il n'est pas évident dans la structure de données que le sous-ensemble est un stable, à vérifier.
- Idem maxClique
- Pour la coloration, un encodeur peut être l'affectation sommets-couleurs. Là encore, la structure d'encodage ne donne pas directement la réalisation des contraintes.

Plan

Heuristiques constructives

En général, construire une solution réalisable à un PLNE est un problème NP-complet.

Sur des problèmes spécifiques, il est trivial de construire des solutions réalisables (ex : TSP, p-median, stable, coloration)

Enjeu : pouvoir trouver rapidement des solutions d'assez bonne qualité.

Heuristiques constructives :

- Algorithme glouton
- GRASP, algorithme glouton randomisé
- Construction par décomposition (CMSA).
- Recherche par faisceaux, "Beam Search"

Heuristique constructive facile : cas p-median

- Etape 1 : Choix aléatoire des p centres de cluster.
- Implémentation : on tire aléatoirement pour chaque point un réel entre 0 et 1, on considère les points donnant les p plus petites valeurs.
- Etape 2 : Affectation des N points au cluster du centre le plus proche
- Complexité du calcul : $O(pN)$ pour construction solution et calcul de la fonction objectif en $O(N)$.

Heuristique constructive aléatoire : cas maxStable

- maxStable : Choix aléatoire d'un sous ensemble de $[\![1, N]\!]$.
- Implémentation : on tire aléatoirement pour chaque point un binaire, donne l'appartenance au sous ensemble choisi.
- Complexité du calcul : $O(N)$ pour définir sous-ensemble et calcul du cardinal.
- Compter sur la chance pour avoir une solution vérifiant les contraintes de stable ou de clique, très peu probable en général..
- Variante respectant les contraintes : à chaque tirage, on vérifie si le candidat à ajouter n'est pas relié à un des sommets déjà sélectionnés. On le sélectionne uniquement si la solution courante est un stable. Invariant de boucle : on a toujours un stable avec les sommets sélectionnés.

Heuristique constructive aléatoire : cas TSP-TSPTW

- TSP : Choix aléatoire d'une permutation de $[\![2, N]\!]$.
- Implémentation : on tire aléatoirement pour chaque point un réel entre 0 et 1, la permutation est donnée par l'ordre croissant sur les valeurs aléatoires.
- Complexité du calcul : $O(N \log N)$ pour définir une permutation, et calcul de la fonction objectif en $O(N)$.
- Extension TSPTW : compter sur la chance pour avoir une solution vérifiant les contraintes de TW, peu probable en général..

Premier bilan

- Il est bien plus aisé d'avoir un encodage qui satisfait directement les contraintes comme p-median.
- La gestion des contraintes supplémentaires, non incluses dans un encodage, apportent des complications avec une heuristique.

Algorithmes gloutons

Algorithmes gloutons :

- De manière générale : construction itérative d'une solution suivant une optimisation locale sans jamais remettre en cause une décision précédemment fixée.
- Algorithme glouton le plus simple : on prend les décisions selon l'ordre d'un critère initialement choisi.
- Exemples Sac à dos : mettre dans le sac à dos l'objet avec le meilleur rapport coût/volume parmi les objets pouvant rentrer
- Glouton adaptatif : le critère de choix local est adapté au cours des itérations.
- N.B : il existe des problèmes d'optimisation où des algorithmes gloutons fournissent des solutions optimales : sac à dos continu, arbre couvrant de poids minimum . . .cf théorie des matroïdes

`https://www.gerad.ca/~alainh/Matroides.pdf`

`https://www.lirmm.fr/~gioan/telecharger/2013%20Gioan%20Ramirez-Alfonsin%20-%20Cours%20sur%20les%20matroides%20(orientes)%20-%20GDRIM2013.pdf`

Algorithme glouton pour stableMax

- Quels éléments a t'on a priori dans un stable ?
- Des noeuds de faible degrés !
- On calcule le tableau des degrés $O(|E|)$, et on trie initialement les sommets par degré minimal. $O(|V| \log |V|)$
- On parcourt les sommets dans cet ordre. La liste I des stables est initialement vide.
- Si un sommet considéré peut être ajouté dans I, i.e., voisin d'aucun sommet de I, on l'ajoute dans I.
- Une fois tous les sommets parcourus, on renvoie les sommets de I.
- Heuristique gloutonne en $O(|E| + |V| \log |V|)$

Algorithme glouton adaptatif pour stableMax

- Dans l'algorithme précédent, le critère de choix des sommets à ajouter est uniquement basé sur le tri initial des degrés
- Version adaptative : on recalcule les "degrés induits".
- Une fois un sommet v choisi pour le stable, cela revient à considérer un stable dans le graphe où on a retiré les voisins de v.
- Approche "destructive" : on recopie le graphe $G = (V', E')$ (ou un système de marqueurs pour sommets et arrêtes éliminés). On stocke le stable courant dans une liste I, initialement vide.
- Tant que $V' \neq \emptyset$:
  - On considère un sommet v de V' de degré minimal dans (V', E').
  - On ajoute v dans I et on actualise (V', E') en enlevant le sommet v et ses voisins et les arrêtes contenant v et ses voisins.
- A la fin, on renvoie les sommets de I, un stable.
- Heuristique gloutonne en $O(|V||E| + |V|^2 \log |V|)$, mais on a adapté les choix gloutons aux perturbations induites par les choix précédents.

Problème de coloration des sommets, heuristiques

De nombreuses heuristiques ont été étudiées pour résoudre le problème coloration des sommets d'un graphe de manière heuristique, l'état de l'art est conséquent sur ce pb très étudié.

On peut définir assez naturellement des constructions heuristiques de type gloutonnes.

Diverses méta-heuristiques classiques ont été étudiées sur ce problème, état de l'art récent et encore très actif :

Brélaz, D. (1979). New methods to color the vertices of a graph. Communications of the ACM, 22(4), 251-256.
Laguna, M., & Martí, R. (2001). A GRASP for coloring sparse graphs. Computational optimization and applications, 19(2), 165-178.
Galinier, P., Hamiez, J. P., Hao, J. K., & Porumbel, D. (2013). Recent advances in graph vertex coloring. Handbook of optimization, 505-528.
Jin, Y., Hamiez, J. P., & Hao, J. K. (2017). Algorithms for the minimum sum coloring problem : a review. Artificial Intelligence Review, 47(3), 367-394.
Mostafaie, T., Khiyabani, F. M., & Navimipour, N. J. (2020). A systematic study on meta-heuristic approaches for solving the graph coloring problem. Computers & Operations Research, 120, 104850.

Heuristique de Welsh et Powell

1. Calculer le degré de chaque sommet.
2. Trier les sommets par ordre de degrés décroissants.
3. Attribuer au premier sommet (A) de la liste une couleur.
4. Suivre la liste en attribuant la même couleur au premier sommet (B) qui ne soit pas adjacent à (A).
5. Suivre (si possible) la liste jusqu'au prochain sommet (C) qui ne soit adjacent ni à A ni à B.
6. Continuer jusqu'à ce que la liste soit finie.
7. Prendre une autre couleur pour le premier sommet (D) non encore coloré de la liste (s'il en reste) et répéter les opérations 3 à 7 tant que tous les sommets ne sont pas colorés.

Cette méthode peut aboutir à des piètres colorations. Si G est une couronne à 2n sommets, on peut avoir une coloration utilisant n couleurs, alors que le nombre chromatique est 2.

D. J. A. Welsh, M. B. Powell. An upper bound for the chromatic number of a graph and its application to 565 timetabling problems, The Computer Journal 10 (1) : pp. 85–86, (1967).

Algorithme glouton adaptatif pour le TSP

- La ville 1 est le point de départ.
- On choisit pour ville suivante, la ville la plus proche de 1, disons v_2.
- Pour choisir la troisième ville, on choisit la ville la plus proche de v_2 autre que $1 = v_1$
- On itère ...
- Une fois les villes $v_1, \ldots, v_k$ choisies, la ville suivante v_{k+1} est la ville la plus proche de v_k autre que $v_1, \ldots, v_{k-1}$

Algorithme glouton adaptatif pour le TSPTW

- En appliquant le glouton du TSP, il faut encore compter sur la chance pour avoir une solution réalisable . . .
- Autre critère glouton : considérer la ville qui a l'heure de fin la plus basse (pour essayer d'assurer la faisabilité)
- Autre critère glouton : considérer la ville atteignable sans attente qui a l'heure de fin la plus basse (pour essayer d'assurer la faisabilité)
- Autre critère glouton : considérer la ville qui minimise le critère (heure maximale de fin) - (heure de début de livraison en atteignant directement la ville)
- On peut construire une fonction de score, somme pondérée de plusieurs critères : critère glouton pour coût (ville plus proche) et critères pour augmenter les chances d'avoir des solutions réalisables.

$\Longrightarrow$ Illustre des difficultés en heuristiques lorsqu'on ajoute des contraintes, et qu'un algorithme glouton peut être modifié pour mieux prendre en compte des contraintes

Algorithme glouton randomisé : GRASP

- Extension GRASP : Glouton adaptatif avec aléatoire.
- Phase gloutonne randomisée : on prend aléatoirement une des toutes meilleures de l'optimisation locale à chaque itération.
- Paramètre glouton randomisé : loi de probabilité, et nombre de meilleure solutions potentielles pouvant être considérées
- "multi-start" : phase gloutonne randomisée peut être répétée pour fournir des solutions différentes, permet de prendre la meilleure solution (ou d'avoir plus de chances d'éviter des infaisabilités de contraintes)
- GRASP : pour chaque solution construite par glouton randomisé, on cherche à améliorer localement la solution obtenue (recherche locale, cf section suivante).

Resende, M. G., & Ribeiro, C. C. (2014). GRASP : Greedy randomized adaptive search procedures. In Search methodologies (pp. 287-312). Springer, Boston, MA.

Construction heuristique par décomposition

- Pour des grandes instances, on aurait envie de résoudre des problèmes plus petits ou plus facile à résoudre et de les recoller pour former une solution réalisable du grand problème.
- Phase décomposition : au cas par cas, suivant structure du problème.
- Algos de clustering peuvent être utiles pour des données localisées (ex TSP).
- Résolution de sous-problèmes indépendants d'une taille définie (par méthode exacte ou heuristique constructive)
- Résolution de sous-problèmes indépendants : peut être implémenté en parallèle.
- Recoller les solutions : un autre petit problème d'optimisation
- CMSA, Construct Merge Solve and Adapt : on ajoute de l'amélioration par recherche locale

Construction par décomposition : ex TSP

- On partitionne l'ensemble des villes en ensembles de cardinal N maximums par similarité, définissant un nombre de clusters $\lceil n/N \rceil$. (de bonnes heuristiques de clustering pour cela)
- Pour chaque cluster de villes, on calcule indépendamment (implémentation parallèle) :
  - les deux villes les plus éloignées du cluster (algo naïf en $O(N^2)$)
  - Un court chemin entre ces deux villes passant par tous les points (heuristiques ou algo exact, N est paramétré pour cela)
- On retrouve une solution du TSP en reliant les 2 $\lceil n/N \rceil$ villes en cherchant à minimiser la distance à rajouter.

TSP : Algorithme d'approximation de Christofides

- Cas du TSP avec distances respectant l'inégalité triangulaire (hypothèse réaliste)
- A partir du graphe complet formé par les villes, avec comme poids les sommets, on peut calculer l'arbre couvrant de poids minimal (algo Prim ou Kruskal, en temps $O(N^2)$)
- On répare l'arbre couvrant en cycle Hamiltonien, opération en $O(N)$, ça donne même une 2-approximation.
- Amélioration dans l'algorithme de Christofides : réparation avec un algorithme de couplage parfait (perfect matching), donne même une 3/2-approximation, ie au plus 50% de sur-coût par rapport à l'optimum dans le pire cas (bien mieux en pratique).

`http://www.cs.cornell.edu/courses/cs681/2007fa/Handouts/christofides.pdf`

`https://pypi.org/project/Christofides/`

`https://github.com/sth144/christofides-algorithm-cpp`

Construction par décomposition : ex maxStable

- Comment décomposer un pb max stable ?
- Sur des graphes à plusieurs composantes connexes, OK, les résultats s'ajoutent.
- Si on enlève des sommets (et leurs arêtes) : on garde les solutions stables ne contenant pas les points enlevés
- Pour décomposer : enlever des noeuds pour avoir plusieurs composantes connexes.
- Comment décomposer à peu de frais ? Par exemple, utiliser Min cut (dual Max-Flow), et enlever les sommets de la coupe.
- RQ : résolution polynomiale exacte de Min cut (dual Max-Flow), algo de Karp-Ford-Fulkerson pour Max-Flow.

Construction par décomposition : ex maxStable

- On itère en utilisant Min cut pour enlever (désactiver) des sommets et avoir des composantes connexes de taille au plus N
- Calcul exact ou construction heuristique de plus grands stables sur les composantes connexes.
- En recollant, on obtient un stable du graphe initial.
- On peut essayer d'ajouter des sommets (manière gloutonne ou autre) retirés par décomposition heuristique.

Algorithme "Beam Search"

- Beam Search : algorithme de recherche en faisceau
- Idée : recherche arborescente est exponentielle (B&B, PPC), limiter le parcours en largeur à k noeuds, k paramètre.
- Choix des k noeuds : selon un critère glouton ou de borne.
- Partant de k noeuds, on considère les k meilleurs noeuds fils suivant le critère choisi précédemment.
- Variante : on peut garder une part d'aléatoire (comme GRASP) dans les solutions retenues
- N.B : peut être codé génériquement sur un algorithme de recherche arborescente Branch&Bound.
- Beam Search : méthode utilisée en IA pour limiter un parcours arborescent
- Beam Search : algorithme très efficace pour des problèmes fortement contraints (ex : Challenge ROADEF 2018)

Algorithme "Beam Search" : ex TSP

- Le noeud 1 est le premier choisi
- Initialisation : on choisit les k villes les plus proches de la ville 1, initialisation avec k listes de 2 villes.
- Pour chacun des k noeuds, on calcule toutes les distances en rajoutant une ville non incluse dans la sous-énumération du noeud. $O(Nk)$ sous noeuds énumérés, étape parallélisable.
- On considère les k meilleurs nouveaux noeuds fils, (ou $k' < k$ meilleurs et $k - k'$ choisis aléatoirement).
- On itère jusqu'à avoir des solutions réalisables.
- N.B : schéma permet d'élaguer des sous-énumération impossibles pour TSPTW, peut aussi être amélioré en détectant les conflits au plus tôt comme en PPC.

Heuristiques constructives et parallélisation (1)

- Les heuristiques gloutonnes et Beam Search opèrent des opérations séquentielles, la parallélisation ne peut être effectuée qu'à l'intérieur de ces opérations.
- Pour une heuristique de décomposition, on peut traiter des sous-problèmes indépendamment, pour une implémentation parallèle.
- GRASP : l'aspect "multi-start" permet une parallélisation de plus haut-niveau, pour chaque processus (ou thread), une initialisation différente.

Heuristiques constructives et parallélisation haut niveau(2)

- Autre idée : on utilise un “porte-feuille” de stratégies heuristiques, des heuristiques différentes ou des paramétrisations différentes
- On effectue les calculs en parallèles et on prend la meilleure solution.
- Intérêt : augmente la robustesse de l'approche sur différents types d'instances, où différentes propriétés numériques interviennent.
- Pour des problèmes fortement contraints, il n'est pas forcément facile de construire une solution réalisable, la parallélisation augmente les chances d'avoir une solution réalisable !

Parallélisation de stratégies constructives, ex TSPTW

- Stratégie 1 : glouton ou GRASP avec minimisation locale de distance,
- Stratégie 2 : glouton ou GRASP avec choix local pour minimiser l'"urgence", on va à la ville où le travail doit être terminé le plus rapidement, pour avoir le plus de chances d'avoir une solution réalisable.
- Stratégie 3 : on minimise les temps d'attente avec les débuts de TW.
- Stratégie 4 : critère agrégé prenant en compte les stratégies 1,2,3 avec différentes pondérations.
- L'efficacité varie selon les propriétés numériques des instances.
- A priori, 1 est une bonne stratégie si les fenêtres de temps ne sont pas trop contraignantes, 2 et 3 essayent d'assurer d'avoir une solution réalisable.
- On peut faire une combinaison de ces stratégies à l'intérieur d'une heuristique.
- La parallélisation en prenant des stratégies très différentes peut permettre d'augmenter la robustesse de la construction heuristique pour trouver des solutions réalisables et de bonne qualité.

Parallélisation avec OpenMP

- On part d'un porte-feuille d'heuristiques constructives. On dispose d'une heuristique paramétrique pour un problème d'optimisation combinatoire tel que le problème de sac à dos, de signature :
- `void heurConstr(vector<int> & sol, float& cost, int seed)`
- `heurConstr` prend en argument une solution initiale encodée comme un vector de `int`, un mode/paramètre de résolution `seed`. `heurConstr` remplit le vector sol avec l'encodage de la meilleure solution trouvée, et le flottant `cost` avec le coût de la solution trouvée .
- On veut lancer des calculs indépendants sur plusieurs threads et que la meilleure solution soit écrite dans un fichier. Facile avec OpenMP, on a tout en mémoire.

$\Longrightarrow$ Comment faire la même chose avec MPI ?

```
#include <mpi.h>
#include <vector>

int main(int argc, char *argv[]) {
  int rank, size,bestRank;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Status s;
  vector<int>  sol;
  float cost=0;

  heurConstr(sol, cost, rank);

  if (rank != 0) MPI_Send(&cost, 1, MPI_FLOAT,0,j, MPI_COMM_WORLD);

  if (rank == 0) {
      vector<float> solVal;
      solVal.push_back(cost);
      for(int j=1;j<size; j++){
            MPI_Recv(&cost, 1, MPI_INT, 0,j, MPI_COMM_WORLD, &s);
            solVal.push_back(cost);
            }
       vector<float>::iterator bestVal = max_element(solVal.begin(), solVal.end());
       bestRank = distance(solVal.begin(), bestVal);
      }

  MPI_Bcast(&bestRank, 1, MPI_INT, 0, MPI_COMM_WORLD);

  if (rank == bestRank)  /* print solution in file */

  MPI_Finalize();
  return 0;
}
```

Parallélisation avec MPI, et réduction MAXLOC

- Première version : chaque processus envoie la valeur de sa solution à un même processus (0 par exemple), le processus 0 détermine quel processus a la meilleure solution, cet indice est diffusé pour que l'indice correspondant puisse afficher/écrire la solution complète.
- Bcp de communications avec cette première version !
- Variante : On peut faire la même chose avec une unique communication collective : un Allreduce avec MINLOC ou MAXLOC pour récupérer la meilleure solution en valeur et l'indice correspondant.

`https://www.mpi-forum.org/docs/mpi-1.1/mpi-11-html/node79.html`

Version avec MAXLOC

```
#include <mpi.h>
#include <vector>

int main(int argc, char *argv[]) {
  int rank, size,bestRank;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  vector<int>  sol;
  float cost=0;

   heurConstr(sol, cost, rank);

   struct {
        float val;
        int   id;
   } in, out;

  in.id = rank;
  in.val = cost;

  MPI_Allreduce(in, out, 1, MPI_FLOAT_INT, MPI_MAXLOC, MPI_COMM_WORLD);

  if (rank == out.id)  /* print solution in file */

  MPI_Finalize();
  return 0;
}
```

Plan

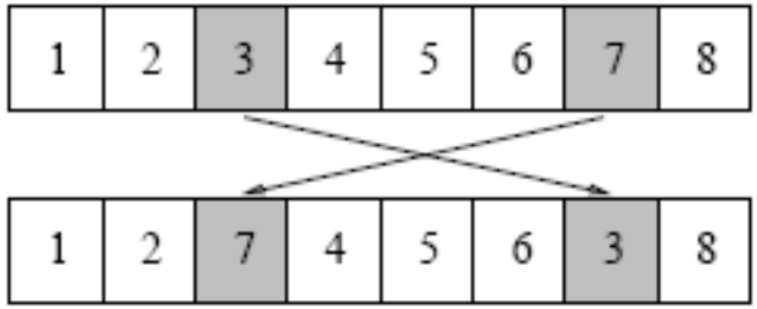
(Méta-)Heuristiques perturbatives de trajectoire

- Heuristique perturbative/recherche locale : on passe d'une solution à une autre par une petite modification locale, pour espérer au cours des itérations (avec un critère d'arrêt défini), avoir parcouru (ou convergé) vers une bonne solution
- Premier point : comment définir une modification locale ?
- On part d'une solution réalisable : utilise heuristique constructive en amont.
- critère d'arrêt : temps total défini, nombre d'itérations, ...
- trajectoire : par opposition aux populations

Voisinages, minimums locaux/globaux

- L'encodage définit un espace X de toutes les solutions réalisables.
- Un voisinage est une fonction $\mathcal{N}$ qui associe un sous-ensemble de X à toute solution $x \in X$. Une solution $x_0 \in \mathcal{N}(x)$ est dite voisine de x.
- Les voisinages dépendent du problème : point laissé générique lors de la définition d'une méta-heuristique, à spécifier.
- Pour un problème donné, de multiples voisinages peuvent être définis en général.

Exemple TSP : voisinage d'échange



voisinage d'échange : $\mathcal{N}(x)$ est l'ensemble des permutations atteignables à partir d'une permutation x de $[\![1; N]\!]$ en conservant au $N-2$ positions de villes définies par x

$|\mathcal{N}(x)| = N(N-1)/2$

Généralisation k-échanges (aussi noté k-opt) : on conserve au moins $N-k$ positions pour définir $\mathcal{N}_k^{swap}(x)$

$$|\mathcal{N}_k^{swap}(x)| = \frac{N!}{k!(N-k)!} = \Theta(N^k)$$

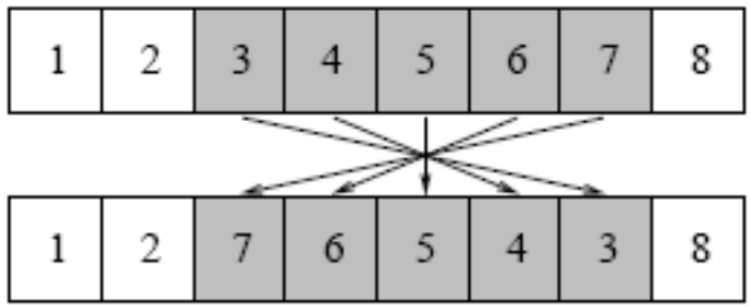
Exemple TSP : voisinage d'insertion

voisinage d'insertion : $\mathcal{N}(x)$ est l'ensemble des permutations atteignables à partir d'une permutation x de $[\![1; N]\!]$ en conservant $N - 1$ ordres entre villes définies par x

$|\mathcal{N}(x)| = O(N^2)$

On peut étendre aussi à un ordre supérieur

Exemple TSP : voisinage d'inversion



voisinage d'inversion : $\mathcal{N}(x)$ est l'ensemble des permutations atteignables à partir d'une permutation x de $[\![1; N]\!]$ en appliquant un miroir à une sous permutation d'un ensemble $[\![a; b]\!]$

$|\mathcal{N}(x)| = N(N-1)/2$

Voisinages pour le p median

- Encodage du p-median : x un vecteur de N booléens de somme p
- Sur un tel encodage, on peut définir la distance de Hamming : nombre de valeurs différentes entre deux vecteurs :

$$d^{Hamming}(x, y) = \sum_{i=1}^{N} |x_i - y_i|$$

- $\mathcal{N}_1(x)$: ensemble des vecteurs y de N booléens de somme p, tels que $d^{Hamming}(x, y) \leqslant 2$
- $\mathcal{N}_m(x)$: ensemble des vecteurs y de N booléens de somme p, tels que $d^{Hamming}(x, y) \leqslant 2m$

$$|\mathcal{N}_1(x)| = p(N - p) \leqslant \frac{1}{4}N^2 = O(N^2)$$

Voisinages, exemple maxStable

Pour maxStable, l'encodage est également défini par un vecteur de $N = |V|$ booléens

On peut également définir des voisinages avec la distance de Hamming

Il faut également prendre en compte que le voisin doit aussi définir un stable

Voisinages et minimums locaux

- L'encodage définit un espace X de toutes les solutions réalisables.
- Soit f la fonction objectif de l'optimisation définie pour $x \in X$.
- Une solution $x \in X$ est un minimum local relativement à la structure de voisinage $\mathcal{N}$ si $f(x) \leqslant f(x_0)$ pour tout $x_0 \in \mathcal{N}(x)$.
- $x \in X$ est un minimum global si $f(x) \leqslant f(x_0), \forall x_0 \in X$
- Un minimum global est minimum local quelque soit le voisinage $\mathcal{N}$ considéré.
- Réciproquement, il existe en général des minimums locaux non globaux.

$\Longrightarrow$ Extension des notions d'optimisation continue, d'optimums locaux ou globaux

Rappel : Minimum local/global en continu

Soit $f : A \to \mathbb{R}$. On dit que

x_0 est un minimum local si

$$\exists \varepsilon > 0 \; \forall x \in]x_0 - \varepsilon, x_0 + \varepsilon[\cap A, \; f(x_0) \leqslant f(x)$$

x_0 est un minimum global si

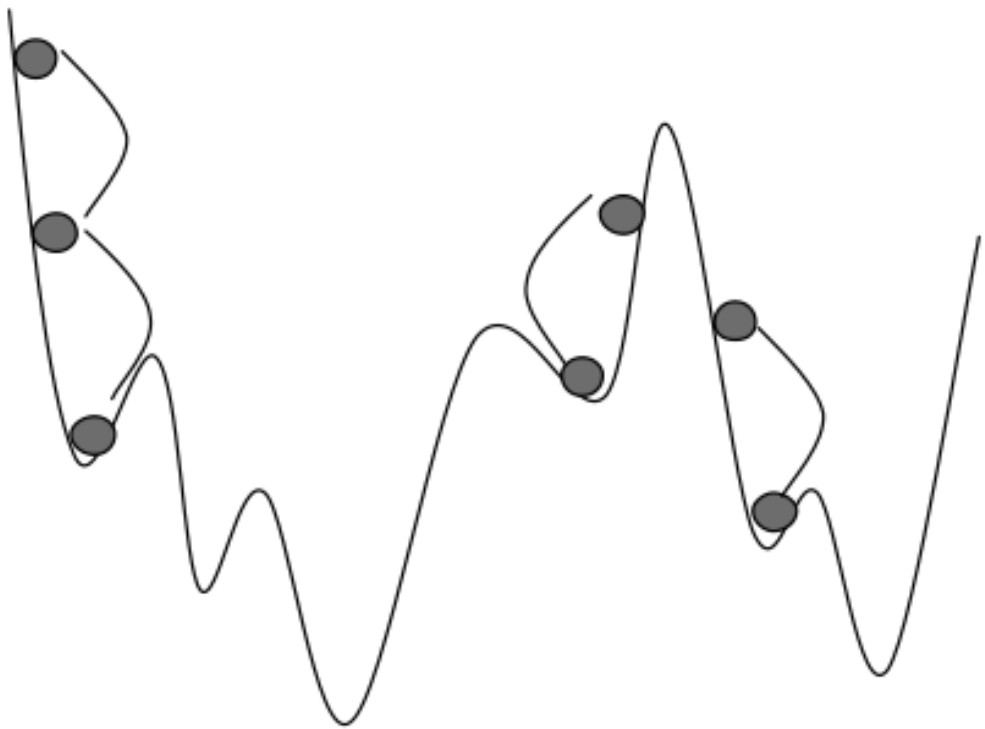
$$\forall x \in A, \; f(x_0) \leqslant f(x)$$

Si $f : A \to \mathbb{R}$ est dérivable, et si un point intérieur $x \in \mathring{A}$ est un extremum local, cela implique $f'(x_0) = 0$. (la réciproque est fausse)

Et généralisation $f : A \to \mathbb{R}^n$, avec distance/norme de $\mathbb{R}^n$, f' désignant gradient.

$\Longrightarrow$ Quelle différence majeure sur les propriétés de voisinages continu/discret ?

Illustration, analogie avec le continu



Différence entre voisinages continu/discret

Les voisinages discrets sont uniquement définis, la localité est une notion sur un $x \in]x_0 - \varepsilon, x_0 + \varepsilon[$ avec $\varepsilon > 0$ petit, aussi petit soit il.

Cette notion (et donc la dérivée, les gradients) ne peut être retranscrite en discret.

Dérivée discrète ? OK, l'algo de gradient sera étendu ...

Différence majeure : les voisinages sont uniquement définis en continu, alors qu'en discret, on peut avoir de multiples types de voisinages.

La notion de minimum local est relative à la définition d'un voisinage.

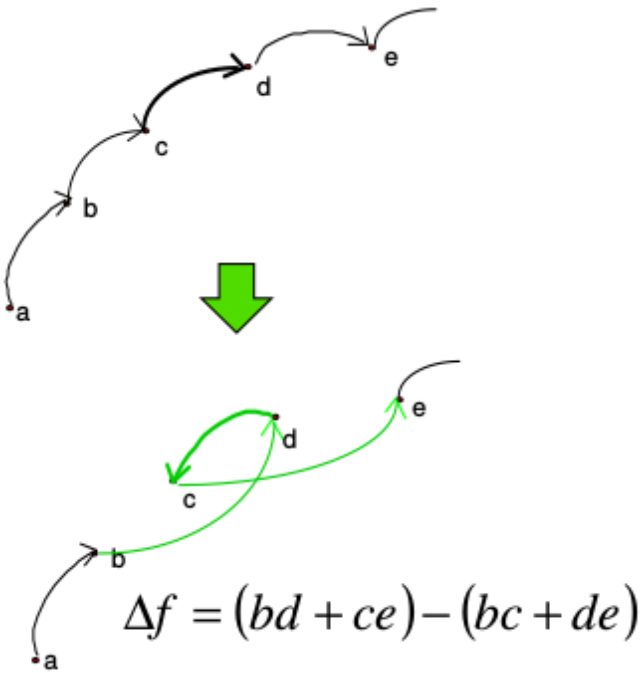
Un minimum local pour un voisinage discret peut ne plus être minimum local en considérant un autre voisinage discret.

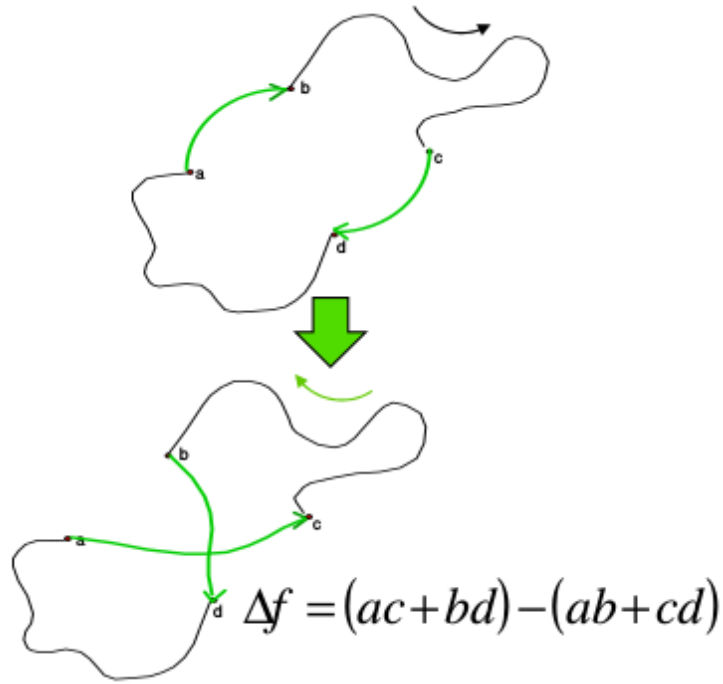
Si $\mathcal{N}_1(x) \subset \mathcal{N}_2(x)$ pour tout $x \in X$, un minimum local pour $\mathcal{N}_2$ l'est aussi pour $\mathcal{N}_1$ (preuve triviale)

Prouver qu'une solution est minimum local

- Soit X un espace d'encodage, f la fonction objectif définie sur X et $\mathcal{N}$ un voisinage.
- Question : comment prouver que $x \in X$ est minimum local
- Vérifier : $f(x) \leqslant f(x_0)$ pour tout $x_0 \in \mathcal{N}(x)$.
- sur les exemples, $\mathcal{N}(x)$ de taille polynomiale, calcul $f(x_0)$ polynomial ...
- On peut mieux faire, pour aller plus vite !

Point clé : calcul du coût de solution dans un voisinage





Calcul rapide du coût de solution par modification :

- A partir d'une solution x, on connaît $f(x)$.
- On calcule $f(x_0)$ à partir de $f(x)$ en considérant uniquement les modifications
- Sur les exemples précédents, $f(x_0)$ se calcule en $O(1)$ à partir de $f(x)$

Exemple TSP : voisinage d'insertion

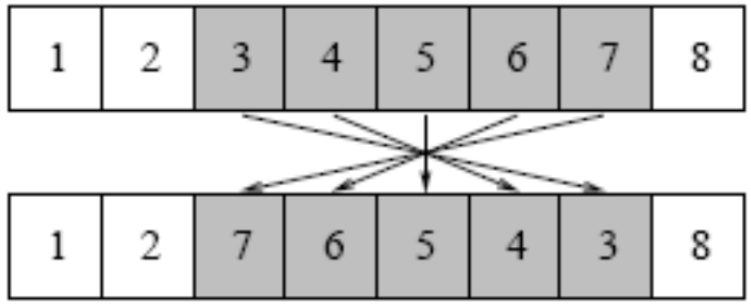
$f(x_0) = f(x) - d_{5,6} - d_{6,7} + d_{1,6} + d_{6,2} + d_{5,7} - d_{1,2}$

$|\mathcal{N}(x)| = N^2$

Prouver que x est un minimum local est en $O(N^2)$

Calcul naïf était en en $O(N^3)$, cela a son importance en pratique !

Exemple TSP : voisinage d'inversion



$f(x_0) = f(x) - d_{2,3} - d_{7,8} + d_{2,7} + d_{3,8}$

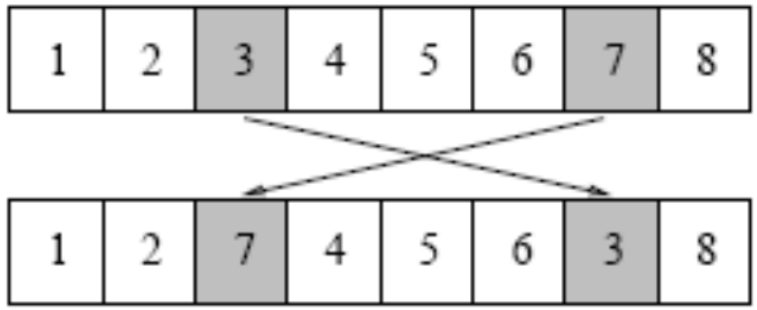
$|\mathcal{N}(x)| = O(N^2)$

Prouver que x est un minimum local est en $O(N^2)$

Calcul naïf était en en $O(N^3)$, cela a son importance en pratique !

N.B : Ici, le calcul de coût est correct uniquement dans le cas symétrique, formule différente dans le cas du TSP asymétrique

Exemple TSP : voisinage d'échange



$$f(x_0) = f(x) - d_{2,3} - d_{3,4} - d_{6,7} - d_{7,8} + d_{2,7} + d_{7,4} + d_{6,3} + d_{3,8}$$

$|\mathcal{N}(x)| = O(N^2)$

Prouver que x est un minimum local est en $O(N^2)$

Calcul naïf était en en $O(N^3)$, cela a son importance en pratique !

Hill climbing : une première recherche locale

- Paramètres : solution initiale $x_i \in X$, on se donne un voisinage $\mathcal{N}$
- Initialisation : $x = x_i$, $f = f(x)$
- TANT QUE (critère d'arrêt temps ou nombre d'itération atteint)
- On parcourt tout le voisinage $\mathcal{N}(x)$ en calculant les $f(x_0)$
- Existe t'il une solution $x_0 \in X$ telle que $f(x_0) < f(x)$?
- Si NON, on sort de la boucle TANT QUE
- Si OUI, on actualise $x = \text{argmin}_{x_0 \in \mathcal{N}(x)} f(x_0)$ et $f = f(x)$.
- FIN TANT QUE
- On renvoie $x, f(x)$

Hill climbing : une recherche locale de descente

- Si le critère d'arrêt de fin d'itération n'est pas atteint, on renvoie un minimum local.
- Chaque itération est améliorante strictement, sinon, on s'arrête.
- Hill Climbing suit la direction de plus grande descente, c'est l'analogue de l'algorithme de gradient.
- Algorithme de recherche locale le plus simple.
- Variantes pour économiser du temps à chaque itération, on peut arrêter l'exploration du voisinage :
  - à la première solution améliorante trouvée ;
  - quand un nombre d'améliorations a été trouvé, et prendre la mileure solution améliorante ;
  - critère d'arrêt avec amélioraion trouvée lié à la complexité. ex : voisinages TSP en $O(N^2)$, on peut arrêter l'exploration au bout de $O(N)$ évaluations si on a trouvé une solution améliorante.

Cas p-median, k-medoids

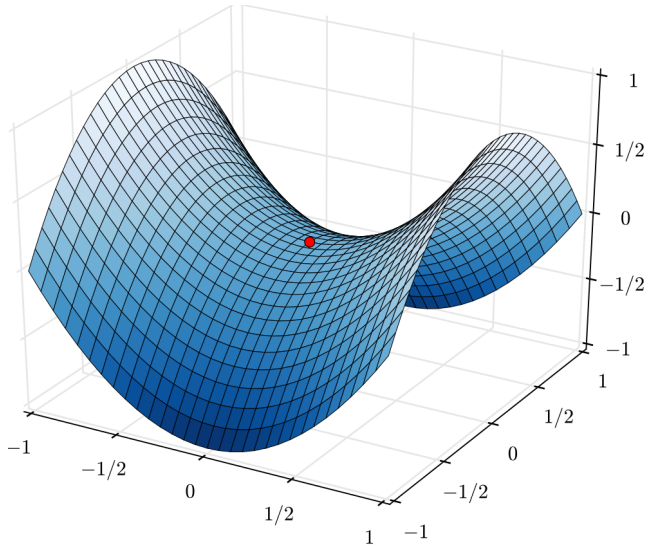
L'algorithme classique pour k-means, k-medoids est une RL hill climbing.

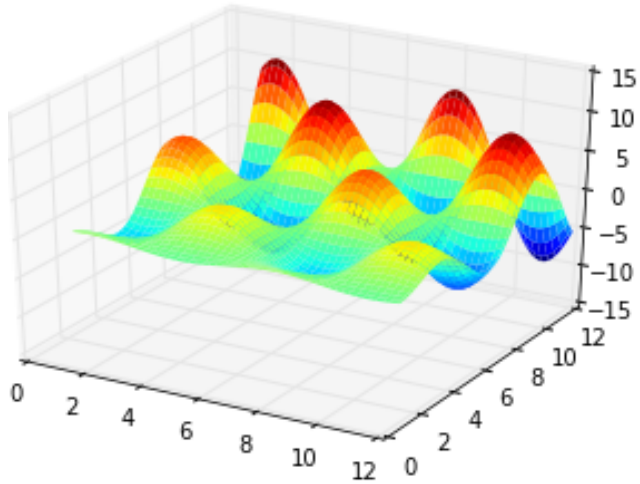
Algorithme suivant pour k-median ($\alpha = 1$) ou k-medoids ($\alpha = 2$) :

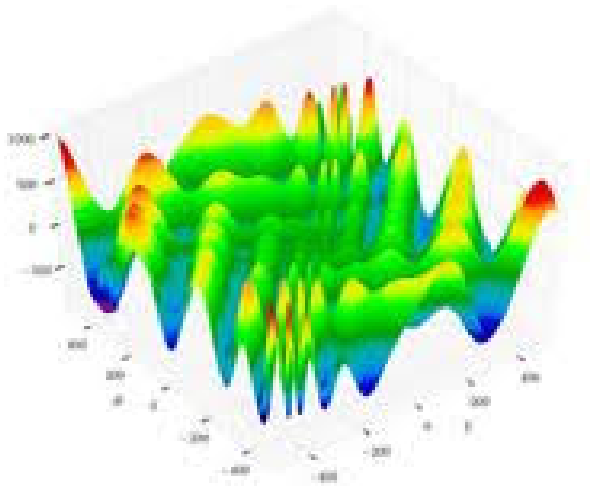
- On part d'un ensemble initial de p centres de clusters.
- Etape 1 : On affecte les points au centre le plus proche.
- Etape 2 : Pour les clusters définis C, on recalcule quel serait le centre idéal de ce cluster : $\text{argmin}_{i \in C} \sum_{j \in C} d_{i,j}^{\alpha}$.
- On itère ce processus, tant que des améliorations sont observés par l'étape 1 ou l'étape 2 (ou fin critère arrêt)

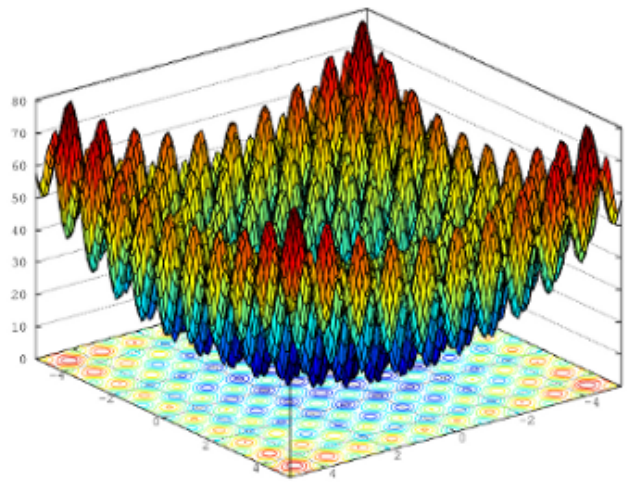
$\Longrightarrow$ Une Hill Climbing, avec une structure de voisinage où des optimisation locales se calculent sans tout explorer le voisinage, par un critère analytique (étape 1).

Illustration de minimums/maximums locaux









$\Longrightarrow$ Même en dimension 2, on peut oublier la notion de monotonie, les tableaux de variation et un faible nombre de minimums/maximums locaux ! !

Sortir d'un minimum local

- En pratique, il existe souvent de multiples minimums locaux, de qualité très variable.
- Hill Climbing peut converger dans un minimum local de mauvaise qualité, le minimum local trouvé dépend de la solution initiale.
- Landscape analysis (analyse de paysages) : analyse statistique des minimas locaux, la localisation des bons minimas locaux et la distance typique entre minimas locaux, donne des indications pour bien paramétrer une heuristique de recherche locale.

$\Longrightarrow$ Dans la suite, différents mécanismes pour sortir d'un minimum local, pour pouvoir explorer d'autres minimums locaux, et des minimas locaux de bonne qualité.

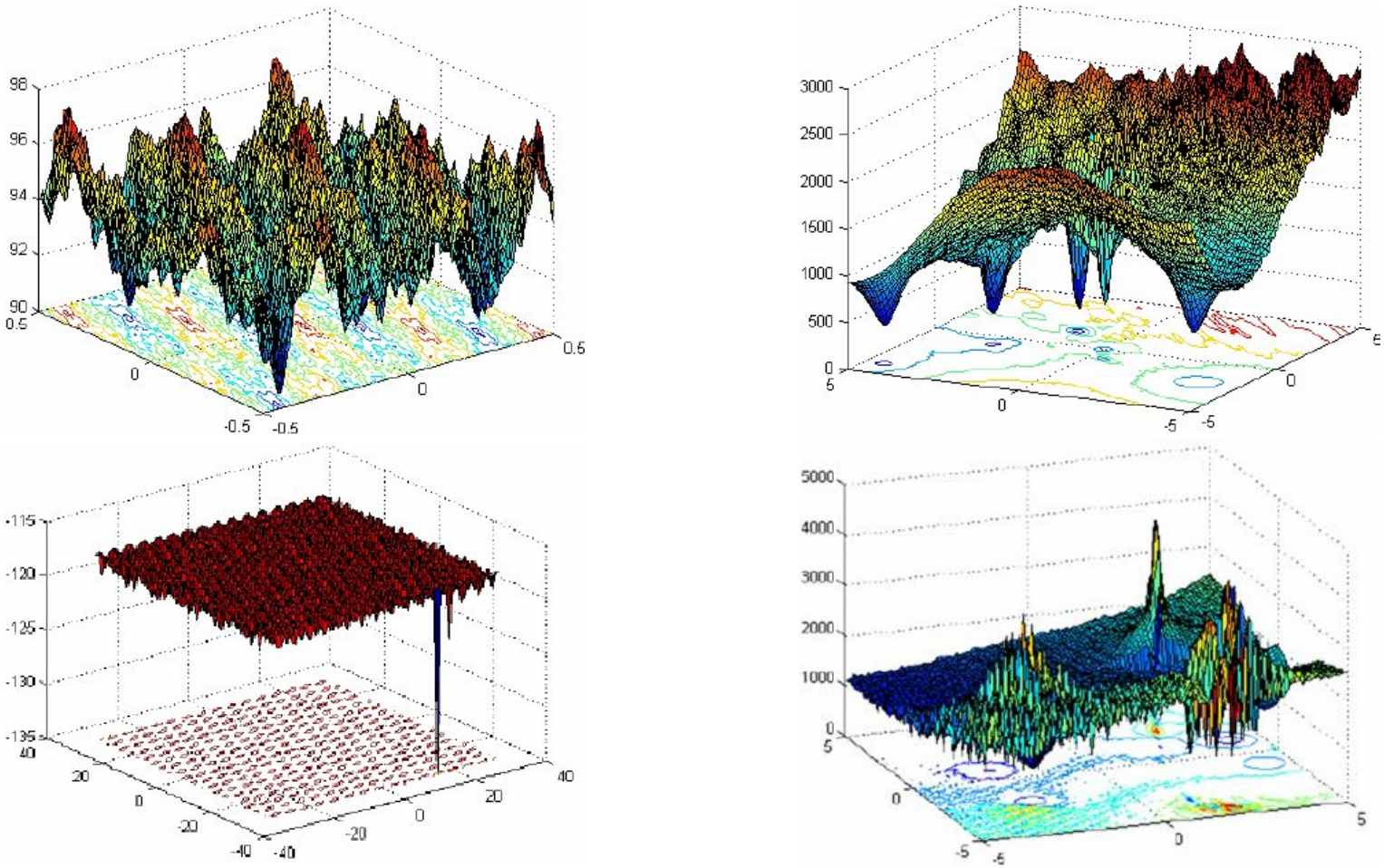
Watson, J. P. (2010). An introduction to fitness landscape analysis and cost models for local search. In Handbook of metaheuristics (pp. 599-623).
`https://cfwebprod.sandia.gov/cfdocs/CompResearch/docs/chapter.pdf`
Humeau, J., Liefooghe, A., Talbi, E. G., & Verel, S. (2013). ParadisEO-MO : From fitness landscape analysis to efficient local search algorithms. Journal of Heuristics, 19(6), 881-915.
`https://hal.inria.fr/hal-00665421v2/document`

Analyse d'un choix de voisinage : landscape analysis

- Rappel : la structure de minimum local est définie par un voisinage.
- Pour Hill Climbing, l'idéal serait d'avoir peu de minimum locaux (à idéalement un, qui serait minimum global.
- L'analyse statistique du nombre de minimas locaux et de leur qualité permet de valider le choix d'un voisinage (plusieurs choix possibles, cf TSP), élément clé de l'efficacité de la RL.
- On peut projeter en dimension 1 : la valeur de la solution courante à chaque itération de recherche locale indique la progression selon un voisinage donné, permet des comparaison de voisinages.
- Projection en dimension 2 existe, pour visualiser, à base de marche aléatoire. Outils disponibles, dont Paradiseo.
- Analyse statistique : partant de solutions initiales aléatoires, quelle est la dispersion des valeurs des minimas locaux ?

`https://www-lisic.univ-littoral.fr/~verel/talks/cec-tuto-all.pdf`
`http://paradiseo.gforge.inria.fr/index.php?n=Doc.TutoMOLesson6`

Landscape analysis projetée en dimension 2



https://www.researchgate.net/publication/270895170_Black_Hole_A_New_Operator_for_Gravitational_Search_Algorithm

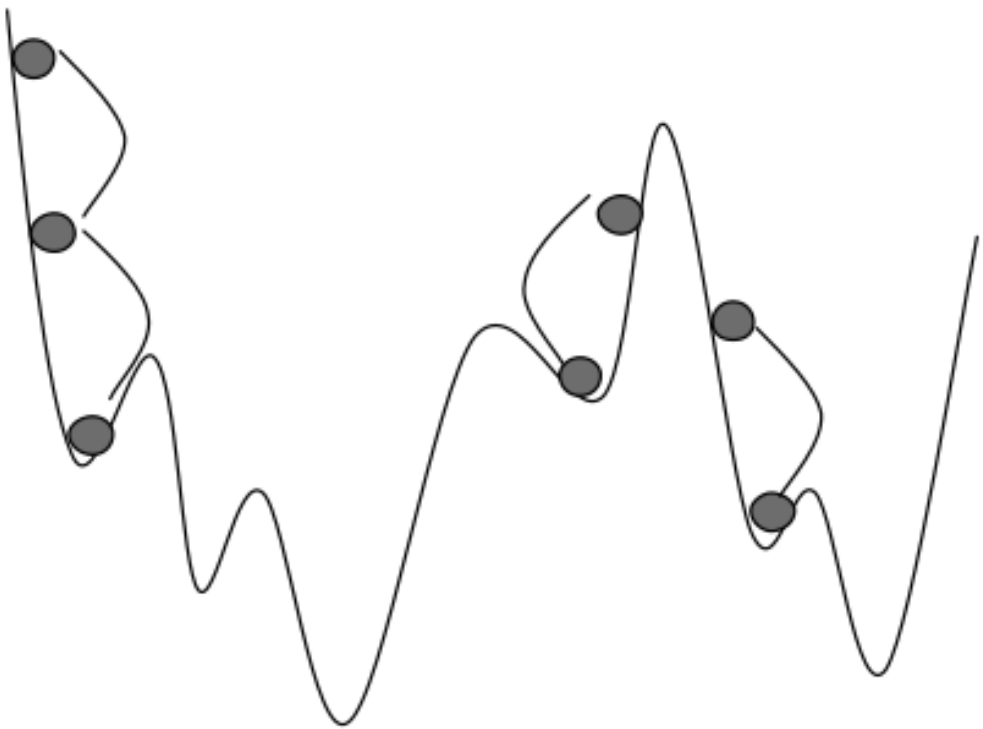
Algorithme ALNS ou VLNS : utiliser de grands voisinages

- ALNS : Adaptative Large Neighborhood Search
- VLNS : Very Large Neighborhood Search
- Si $\mathcal{N}_1(x) \subset \mathcal{N}_2(x), \forall x$, le voisinage $\mathcal{N}_2$ est plus grand que $\mathcal{N}_1$, on prouve facilement que tout minimum local de $\mathcal{N}_2$ est minimum local de $\mathcal{N}_1$.
- Utiliser de plus grands voisinages $\Longrightarrow$ moins de minimums locaux, et de meilleure qualité, qu'en utilisant des petits voisinages.
- Explorer de plus grands voisinages peut demander un temps d'exploration bien plus long.
- Exploration exacte de voisinage de taille exponentielle : algo optimal spécifique (Programmation dynamique souvent), ou algo exponentiel tronqué (B&B) sur une taille spécifique où l'efficacité pratique est connue.
- Une exploration par heuristique est possible, l'optimalité n'est pas théoriquement nécessaire (en pratique, l'heuristique doit être efficace)

Ahuja, R. K., Ergun, Ö., Orlin, J. B., & Punnen, A. P. (2002). A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics, 123(1-3), 75-102.

Pisinger, D., & Ropke, S. (2010). Large neighborhood search. In Handbook of metaheuristics (pp. 399-419). Springer, Boston, MA.

Illustration, analogie avec le continu



Famille VNS : changer de types de voisinages

- MNS : Multi Neighborhood Search
- VNS : Variable Neighborhood Search
- Si $\mathcal{N}_1, \mathcal{N}_2$, sont deux voisinages, ils définissent des minimas locaux. Les minimas locaux communs (intersection) sont les meilleurs minimums locaux (contraposée, un minimum local de l'un mais pas de l'autre permet de définir une chaîne de minimas locaux d'évaluation décroissante vers un minimum local commun).
- Utiliser de plusieurs types voisinages $\Longrightarrow$ moins de minimums locaux, et de meilleure qualité, un minimum local est minimum local de tous les types de voisinages.
- De nombreuses variantes de VNS : VND, Variable Neighborhood Descent, Hill Climbing avec plusieurs voisinages.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. Computers & operations research, 24(11), 1097-1100.

Wu, Q., Hao, J. K., & Glover, F. (2012). Multi-neighborhood tabu search for the maximum weight clique problem. Annals of Operations Research, 196(1), 611-634.

Avoir plusieurs types de voisinages

- Pour utiliser les algorithmes de recherche locale cette section, on a besoin de définir une structure de voisinage.
- Tentation naturelle du fainéant : s'arrêter à la définition d'un unique voisinage, ça suffit pour coder ces algorithmes d'optimisation.
- Avoir des voisinages diversifiés, et de taille variable à un intérêt fort comme présenté avec ALNS et VND pour éviter des mauvais minimas locaux, un point clé de l'efficacité des méta-heuristiques de trajectoires.
- Choisir quel voisinage utiliser peut se faire de manière aléatoire, en définissant des probabilités d'utilisation de types de voisinages.
- On peut regarder l'impact des voisinages, en terme de taux d'acceptation (pourcentage de solutions améliorantes trouvées lorsqu'on utilise un voisinage), ou également chiffrer le rapport amélioration de la valeur de la fonction objectif/ temps passé à explorer le voisinage.
- Stratégie naturelle : utiliser les petits voisinages rapides lorsqu'il est facile d'améliorer la solution courante et les plus gros voisinages avec des temps d'exploration plus long quand les petits voisinages peinent à améliorer
- On adapter les probabilités de choix de voisinages selon l'évolution des taux d'acceptation.

Intensification/Diversification

- Rappel : Les méta-heuristiques visent à explorer l'espace de recherche efficacement afin de trouver des bonnes solutions. Au lieu d'effectuer une recherche exhaustive parfois longue, on n'en explore qu'une partie, guidé par la fonction objectif.
- Intensification : concentration sur une zone précise prometteuse.
- Intensification pure : Hill Climbing, VND.
- Diversification : mécanismes pour une exploration assez large de l'espace de recherche, au minimum, pouvoir sortir d'un minimum local.
- Importance de bien calibrer ces mécanismes antagonistes
- Trop d'Intensification : risque de rester dans même zone, et de rester dans un minimum local.
- Aucune intensification : marche aléatoire, exploration de régions peu prometteuses.

$\Longrightarrow$ pour la suite, on regarde différents mécanismes de diversifications.

Retour GRASP : recherche locale multi-start

- GRASP : Glouton adaptatif avec aléatoire.
- Phase gloutonne randomisée : on prend aléatoirement une des toutes meilleures de l'optimisation locale à chaque itération.
- Phase gloutonne randomisée peut être répétée pour fournir des solutions différentes, permet de prendre la meilleure solution (ou d'avoir plus de chances d'éviter des infaisabilités de contraintes)
- GRASP : pour chaque solution construite par glouton randomisé, on cherche à améliorer par recherche locale (mtnt, on sait ce que c'est).
- RL multistart : on lance une RL à partir de solutions initiales différentes, permet de tomber dans différents minimums locaux, c'est l'initialisation qui conditionne le futur minimum local.

R. Martí, J. Lozano, A. Mendiburu, L. Hernando. Multi-start methods. Handbook of Heuristics,2016.

Resende, M. G., & Ribeiro, C. C. (2014). GRASP : Greedy randomized adaptive search procedures. In Search methodologies (pp. 287-312). Springer, Boston, MA.

Recherche locale/recherche locale itérée

- Recherche Locale Itérée (Iterated Local Search), s'applique pour toute recherche locale (Hill Climbing et les suivantes dans ce cours.)
- Idée : une fois que l'on est dans un minimum local, on perturbe fortement la solution courante, et on relance la recherche locale.
- Permet de sauter d'un minimum local à un autre, et d'explorer plusieurs minimas locaux.
- Ici, le saut se base sur le meilleur minimum local, contrairement à une RL multi-start qui n'utilise pas de telles informations.
- Les ALNS ont une phase dite "destroy" dans "destroy&repair", opérateur de diversification analogue à perturbation ILS.
- Les VNS ont une phase dite "shaking" qui perturbent la solution courante à chaque itération, opérateur de diversification analogue, mais perturbation moins forte en général si souvent répétée.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. In Handbook of metaheuristics (pp. 320-353). Springer, Boston, MA.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2019). Iterated local search : Framework and applications. In Handbook of metaheuristics (pp. 129-168). Springer, Cham.

Opérateurs de perturbation, exemples

- Max Stable : on retire α% des sommets de la meilleure solution courante aléatoirement
- p-median : on sort α% des points choisis comme centres de clusters, et on les remplace par des points qui n'étaient pas des centres auparavant.
- TSP : on retire α% des villes de la solution courante en laissant leur emplacement vide dans l'encodage, et on remplit les trous aléatoirement sans doublon.
- TSP : on retire α% des villes de la solution courante, on considère comme premières villes les villes non retirées et on remplit les derrières villes aléatoirement sans doublon.
- N.B : avec de telles règles dépendant d'un paramètre α, α peut être adaptatif et diminuer au cours du temps.

Recherche taboue : sortir d'un minimum local

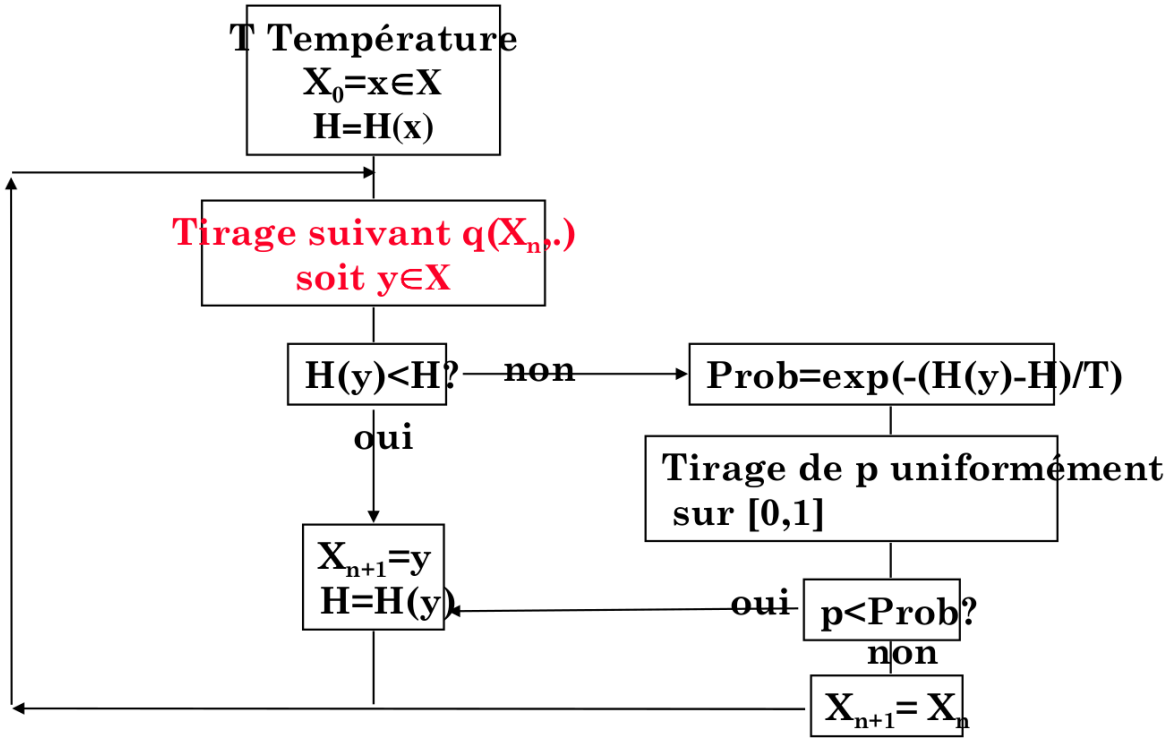
- Version v0 : on définit L une liste taboue qui correspond à un ensemble de points $x \in X$.
- Idée : dans Hill Climbing, on ne parcourt que $\mathcal{N}(x) - L$, on ignore les éléments de la liste taboue.
- Quand on tombe dans un minimum local $x \in X$, on actualise la liste taboue $L = L \cup \{x\}$, et on actualise x un autre point du voisinage.
- Taille maximale de la liste taboue : pour ne pas trop alourdir les calculs, et au bout d'un certain temps, on s'est éloigné des premiers minimas locaux, tabous peuvent être inutiles.
- De nombreuses versions, à adapter selon structure du problème et des voisinages : on peut stocker une information partielle.
- Ex maxStable : stocker uniquement les M éléments de degré minimal, les plus impactants dans le design d'une solution.

Glover, F., & Laguna, M. (1998). Tabu search. In Handbook of combinatorial optimization (pp. 2093-2229). Springer, Boston, MA.

RL sans calcul exhaustif des solutions d'un voisinage

- Calcul exhaustif des solutions d'un voisinage : assez long pour des complexités en $O(N^2)$ et plus.
- Plus rapide : on tire aléatoirement dans le voisinage une solution.
- Au bout de M itérations sans améliorations, on peut faire l'évaluation complète du voisinage (comme Hill Climbing) ou passer à un opérateur de diversification comme si on était dans un minimum local.
- Algorithmes de Metropolis : on accepte une nouvelle solution si elle améliore la solution courante, ou selon une probabilité dépendant de l'écart entre nouvelle solution et solution courante.
- Loi de Boltzmann pour probabilité d'acceptation : À une température donnée T, la probabilité, pour un système physique, de posséder une énergie donnée E, est proportionnelle au facteur de Boltzmann : $\exp\left(-\frac{E}{k_B T}\right)$, où k_B désigne la constante de Boltzmann.

Algorithme de Métropolis



À température intermédiaire, l'algorithme de Metropolis peut autoriser des transformations qui dégradent la fonction objectif pour s'extraire d'un minimum local.

N.B : attention, on considère ici un pb de minimisation, pour passer à un problème de maximisation, il faut être soigneux pour adapter les étapes.

Algorithme de recuit simulé

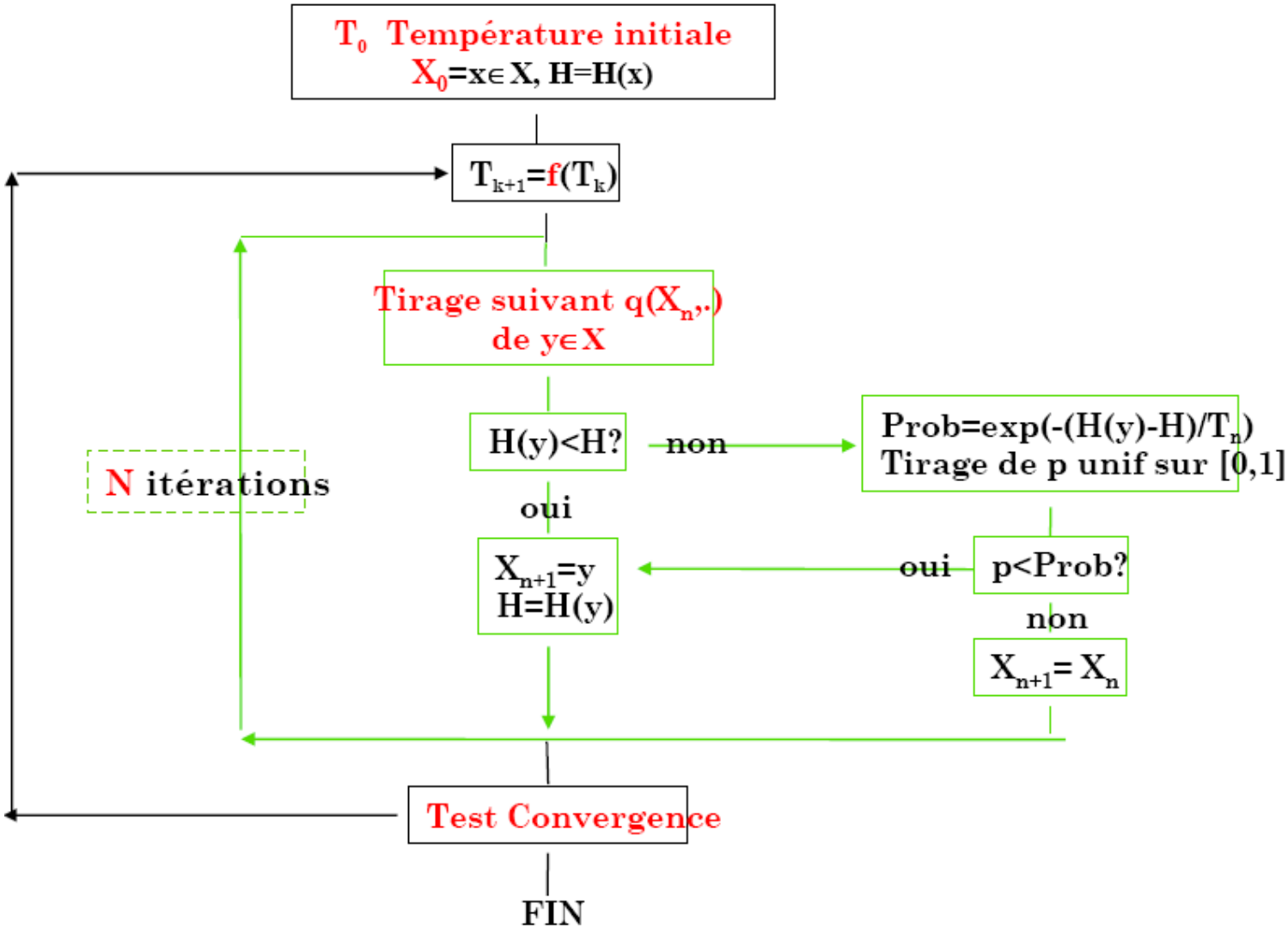
- L'algorithme de recuit simulé date de 1983 : on fait varier la température dans l'algo de Metropolis. (analogie avec le refroidissement en sidérurgie)
- Constat : le refroidissement naturel de certains métaux ne permet pas aux atomes de se placer dans la configuration la plus solide.
- Pour modifier l'état d'un matériau, la température est utilisée comme paramètre de commande.
- La technique du recuit : chauffer le matériau, i.e. lui donner une énergie élevée, puis lent refroidissement.
- Descente en T^o trop rapide $\rightarrow$ des défauts qui peuvent être éliminés par réchauffement local.
- La technique opposée, la trempe : on abaisse très rapidement la température, induit un minimum local de l'énergie.
- Une baisse contrôlée de la T^o conduit à un état stable, un minimum absolu de l'énergie.

S. Kirkpatrick, C.G. Gelatt, M.P. Vecchi, *Optimization by simulated annealing*. Science 220(4598), 671–680 (1983)

Algorithme de recuit simulé, lois de température

- A une itération k, où la solution courante à une valeur H, on actualise la température T_k, et on choisit une solution y aléatoirement dans le voisinage de la solution courante.
- y devient la solution courante si $H(y) \leqslant H$ ou selon un tirage aléatoire de paramètre $p = \exp\left(\frac{H-H(y)}{T_k}\right)$.
- Dans le second critère d'acceptation, le coût de solution est dégradé (opérateur de diversification)
- Les phases de recuits et de descente de température sont paramétrées par la loi d'évolution de T_k.
- A haute température, $\exp\left(\frac{H-H(y)}{T_k}\right) \approx 1$, la plupart des mouvements sont acceptés, l'algorithme équivaut à une marche aléatoire.
- À basse température, $\exp\left(\frac{H-H(y)}{T_k}\right) \approx 0$, la plupart des mouvements augmentant l'énergie sont refusés, l'algorithme se ramène à des itérations "hill climbing".

Algorithme de recuit simulé



Loi de température et convergence

- La généricité de l'algorithme de recuit simulé, sa facilité à le paramétrer, ont contribué à sa popularité, de même qu'un résultat théorique de convergence.
- Le recuit stimulé converge en probabilité vers un optimum global, sous les hypothèses d'ergodicité, avec une loi de température où T_k décroissant au plus vite en $O\left(\frac{1}{\log(k)}\right)$.
- En pratique, la loi logarithmique de décroissance de la température n'est pas utilisée, car inefficace en temps de calcul.

Paramétrisation du recuit simulé

- La température initiale est un paramètre important pour l'efficacité pratique. Critère usuel, considérer ΔH comme un grain raisonnable selon la fonction objectif et prendre une température initiale T_0 tq $\exp\left(-\frac{\Delta H}{T_0}\right) \approx 0.8$.
- En pratique, la loi logarithmique de décroissance ($\frac{1}{T_k} = \frac{1}{T_0} + \beta \log(k+1)$) n'est pas utilisée, car inefficace en temps de calcul. Loi usuelle : $T_k = T_0 \alpha^k$ avec $\alpha < 1$ proche de 1.

$\Longrightarrow$ Un des atouts du recuit simulé, facile à paramétrer, avec peu de paramètres et des règles simples

Recuit simulé et voisinages

- Jusqu'à présent, on tirait aléatoirement un point voisin, la définition du voisinage est assez implicite dans cet opérateur.
- Même intérêt que précédemment à avoir plusieurs types de voisinages.
- Choisir un voisin aléatoirement, c'est choisir d'abord un type de voisinage, et ensuite un élément du voisinage considéré selon une loi uniforme.
- On associe initialement à chaque type de voisinage la probabilité d'être choisi.
- On peut faire varier les probabilités selon le l'évolution du taux d'acceptation : nombre d'améliorations induites par le voisinage/nombre d'essai sur le voisinage
- N.B : concept proche apprentissage par renforcement ϵ-greedy

ILS vs recuit simulé avec loi de température

- Algorithme du kangourou : ILS + SA. (mais élaboré avant ILS selon ses défenseurs)
- Egalement un résultat de convergence pour le kangourou.
- Autre pratique : remonter régulièrement et brutalement la température, courbes $T_k = T_0 \alpha^k$ "par morceaux". Un nouveau paramètre

Stockage de la meilleure solution trouvée

- Hill Climbing et VND : la solution courante est forcément la meilleure solution trouvée.
- Recuit simulé avec décroissance logarithmique : en probabilité, on a convergé presque sûrement vers un optimum.
- En pratique : il vaut mieux stocker la meilleure solution trouvée avec un RL, les mécanismes de diversification peuvent éloigner fortement de la meilleure solution trouvée.

Bilan : méta-heuristiques perturbatives de trajectoire

Algorithme le plus simple : Hill Climbing ;

Variantes pour sortir d'un minimum local :

- Recherche taboue (Tabu Search)
- Recuit simulé (Simulated Annealing)
- Algorithme du kangourou
- Recherche Locale Itérée (Iterated Local Search)
- Recherche à voisinage variables, Variable Neighborhood Search (VNS).
- Adaptative Large Neighborhood Search (ALNS).

N.B : discussions et variantes à propos du recuit simulé discutées précédemment sont valables ou s'adaptent à de nombreuses recherche locales.

Recherche locale et parallélisation

- Hill-Climbing : en évaluant tout un voisinage, on a autant de calculs indépendants que d'éléments dans le voisinage : parallélisation à grain assez fin, OpenMP ou GPU si adapté.
- Avec une recherche locale : plusieurs RL en parallèle en partant de points de départ différent (parallélisation de plus haut niveau, OpenMP et/ou MPI)
- Parallélisation plus bas niveau : dans un recuit simulé, peu d'itérations améliorantes à basse température, plusieurs tests peuvent être effectués en parallèle. Parallélisation à grain assez fin, OpenMP

E. Aarts, F. de Bont, J. Habers, P. van Laarhoven, A parallel statistical cooling algorithm, Lecture Notes in Computer Science, vol 210, 1986, pp 87-97.

P. Roussel-Ragot, P. Siarry, G. Dreyfus, La méthode du "recuit simulé" en électronique : principe et parallélisation, colloque national sur la conception de circuits à la demande, article G2, 1986, pp 1-10.

GRASP et parallélisation

- Parallélisation GRASP : A haut niveau, explorer bcp de solution initiales en parallèle, construction aléatoire différente sur plusieurs processus, phase déterministe
- Parallélisation MPI : ne nécessite pas de communication préalable (graine aléatoire initialisée avec le rang).
- Attention : si les paramètres aléatoires sont initialisés avec la même graine (ou sans graine) : même algo déterministe réalisé sur chacun des processus ...
- Communication pour récupérer meilleure solution de chaque noeud vers un noeud, et diffusion de meilleure solution.

Parallélisation de recherche locale avec MPI

- Un algorithme de recherche locale nécessite des paramètres aléatoires et une solution initiale courante.
- On peut faire tourner indépendamment la RL sur un grand nombre de processus avec MPI, sans communication préalable, et récupérer à la fin la meilleure solution : un Allreduce avec MINLOC ou MAXLOC pour récupérer la meilleure solution.
- On peut synchroniser quelques processus en réactualisant leur meilleure solution trouvée pour des initialisations de RL : Allreduce avec MINLOC ou MAXLOC et Broadcast de solution.
- On peut synchroniser les RL en les arrêtant avec des critères d'arrêt de temps identiques, pour éviter des temps d'attente perdus.
- Globalement : essayer de minimiser le synchronisations, temps perdu à attendre.
- Intérêt à utiliser des communications asynchrones dans un tel cadre.

Recherche locales synchronisées avec MPI

```
#include <mpi.h>
#include <vector>

int main(int argc, char *argv[]) {
  int rank, size,bestRank;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  vector<int>  sol;
  float cost=0;
  int nbIter=5000;
  int nbSync=5;

  heurConstr(sol, cost, rank);
  struct { float val; int  id; } in, out;
  in.id = rank;
  in.val = cost;

  MPI_Allreduce(in, out, 1, MPI_FLOAT_INT, MPI_MAXLOC, MPI_COMM_WORLD);

  for(int i=0; i<nbSync; i++){
    MPI_Bcast(&sol[0], sol.size(), MPI_INT, out.id, MPI_COMM_WORLD);
    heurLocSearch(sol, cost, rank, nbIter);
    in.val = cost;
    MPI_Allreduce(in, out, 1, MPI_FLOAT_INT, MPI_MAXLOC, MPI_COMM_WORLD);
  }

  if (rank == out.id)  /* print solution in file */
  MPI_Finalize();
  return 0;
}
```

Remarques sur la parallélisation précédente

- On peut synchroniser les RL en les arrêtant avec des critères d'arrêt de temps identiques, pour éviter des temps d'attente perdus.
- Globalement : essayer de minimiser le synchronisations, temps perdu à attendre.
- Intérêt à utiliser des communications asynchrones dans un tel cadre.
- Attention : trop de synchronisations vers la meilleure solution courante fait perdre en diversification et augmente le risque d'explorer la même zone autour d'un minimum local.
- On peut définir un sous -ensemble de processus synchronisés, définir un sous communicateurs avec des processus synchronisés, et des processus libres de toute synchronisation.

Méta-Heuristiques parallélisables, des références

Crainic T., and Toulouse M.. "Parallel strategies for meta-heuristics." Handbook of metaheuristics. Springer, Boston, MA, 2003. 475-513.

Talbi, E. and Bachelet V.. "Cosearch : A parallel cooperative metaheuristic." Journal of Mathematical Modelling and Algorithms 5.1 (2006) : 5-22.

Talbi, E. G. (2002). A taxonomy of hybrid metaheuristics. Journal of heuristics, 8(5), 541-564.

Cotta, C., E. G. Talbi, and E. Alba. Parallel Hybrid Metaheuristics. Parallel Metaheuristics : A New Class of Algorithms 4 7 (2005) : 347

Cahon, S. Melab N., and E-G. Talbi. "Paradiseo : A framework for the reusable design of parallel and distributed metaheuristics. Journal of heuristics 10 3 (2004) : 357-380.

Plan

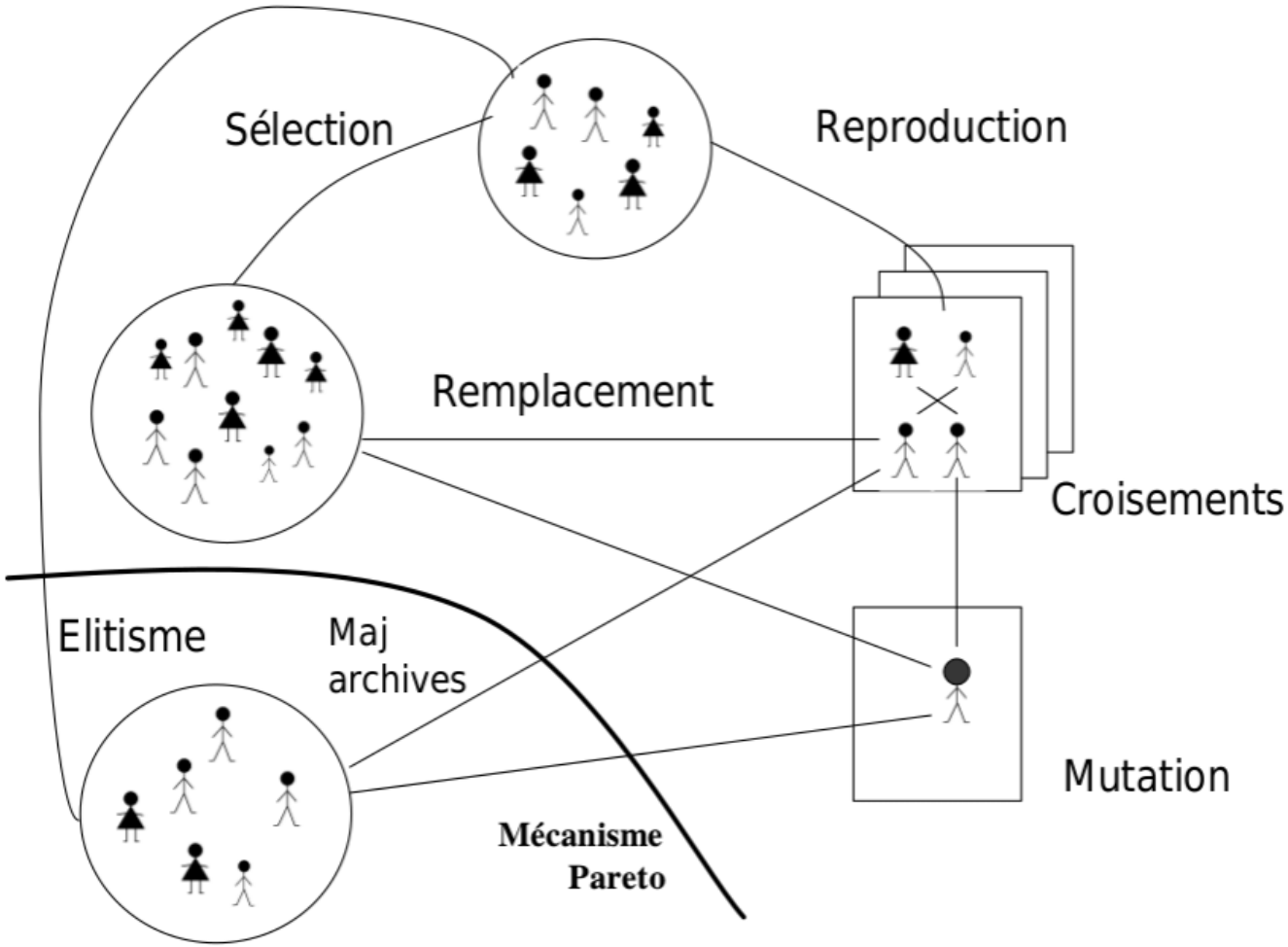
(Méta-)Heuristiques perturbatives de populations

- Heuristique perturbative : nécessite une (des) solution(s) initiales
- Heuristique de population : on traite un pool de solutions, stockés simultanément en mémoire, et on fait évoluer la population dans son ensemble.
- critère d'arrêt : temps total défini, nombre d'itérations, ...
- Exemples :
  - Algorithmes évolutionnaires (Evolutionary Algorithms)
  - Colonies de fourmis (Ant Colony Optimization, ACO)
  - Colonies d'abeilles (Bee Colony Optimization)
  - Essaims particulaires (Swarm Particle Optimization)

Algorithmes évolutionnaires, une méta-heuristique de population

- Construction et évaluation d'une population initiale ;
- Jusqu'à atteindre un critère d'arrêt :
  - Sélection d'une partie de la population,
  - Reproduction des individus sélectionnés,
  - Mutation de la descendance,
  - Evaluation du degré d'adaptation de chaque individu,
  - Remplacement de la population initiale par une nouvelle population.
- N.B : cas particuliers : algorithme génétique (reproduction = cross-over), Algorithmes différentiels (DEA), Scatter Search ...

Algorithme évolutionnaire, illustration



Algorithmes évolutionnaires, opérateurs

- Sélection d'une partie de la population : aléatoire ou autre critère spécifique
- Reproduction des individus sélectionnés : opérateur assez spécifique, fait l'originalité p/r aux heuristiques de trajectoires
- Mutation de la descendance : n'importe quelle recherche locale de trajectoire
- Evaluation du degré d'adaptation de chaque individu : calcul de fonction objectif (facile sur un PLNE, plus difficile quand nécessite des mesures physiques)
- Remplacement de la population initiale par une nouvelle population : aléatoire ou selon évaluation du critère (approche élitiste) ou mixte.

Exemple d'opérateur de reproduction : p-median

- Encodage : vecteur de N booléens indiquant si un point est centre de cluster.
- A partir de deux encodages de solutions, si on sélectionne aléatoirement une valeur, peu de chance d'avoir exactement p centres sélectionnés. (la mutation peut réparer dans une première phase)
- Cross-over respectant la contrainte : à partir des centres définis par deux solutions, on en sélectionne aléatoirement p.

Exemple d'opérateur de reproduction : max Stable

- Encodage : vecteur de N booléens indiquant si un sommet est dans le stable.
- A partir de deux encodages de solutions, si on sélectionne aléatoirement une valeur, très peu de chance d'avoir un stable . . .
- Cross-over respectant la contrainte : à partir des sommets définis par deux solutions, on choisit un sommet aléatoirement, on retire ses voisins, et on itère . . .
- Cross-over respectant la contrainte : à partir des sommets définis par deux solutions, on effectue un calcul exact de stable max sur le sous graphe induit (si assez petite taille).

Exemple d'opérateur de reproduction : TSP

- Encodage : une partition de $\{1, \ldots, N\}$.
- A partir de deux encodages de solutions, si on sélectionne aléatoirement une valeur, très peu de chance d'avoir une partition (pas de doublon ...)
- Cross-over respectant la contrainte : considérer toutes les transitions possibles d'une ville à sa suivante, et algo aléatoire en respectant les contraintes d'élémentarité (pas de doublon)
- Une réparation peut être implémentée à partir de cross over fournissant des solutions non réalisables : fixer les points sans doublons et réparation gloutonne en plaçant les villes non affectées et les doublons une seule fois.

ACO : optimisation par colonies de fourmis.

- ACO, Ant Colony Optimization : optimisation par colonies de fourmis.
- ACO a été introduit en 1996 sur un problème de plus court chemin dans un graphe, et a depuis été étendu comme une méta-heuristique
- Analogie vient de l'observation de l'exploitation des ressources alimentaires chez les fourmis.
- Une colonie de fourmis ayant le choix entre deux chemins d'inégale longueur menant à une source de nourriture a tendance à utiliser le chemin le plus court.
- ACO se base sur la collaboration des individus entre eux, et s'appuie sur le concept d'auto-organisation (ou d'intelligence collective) : un groupe d'individus peu intelligents peut posséder une organisation globale complexe.

Dorigo, M., & Di Caro, G. (1999). Ant colony optimization : a new meta-heuristic. IEEE congress on evolutionary computation-CEC99 (Vol. 2, pp. 1470-1477).

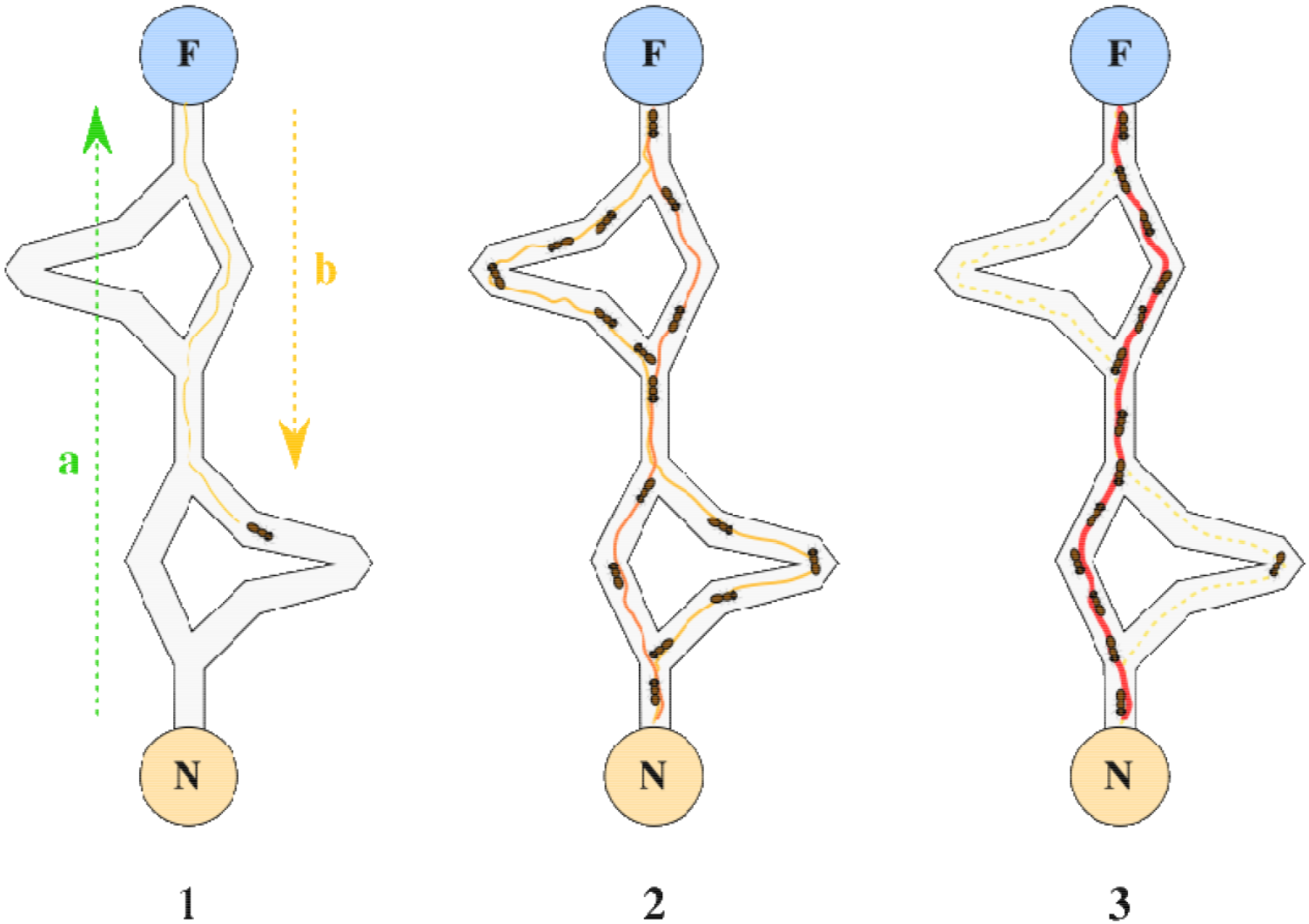
Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. European journal of operational research, 185(3), 1155-1173.

C. Solnon. Optimisation par colonies de fourmis - Collection Programmation par contraintes. Hermès-Lavoisier, 2008

ACO : optimisation par colonies de fourmis

- La méthode ACO construit des solutions à partir des éléments qui ont été explorés par d'autres individus : constructif et une recherche locale.
- Les individus sont des fourmis se déplacent à la recherche de solutions et sécrétent des phéromones, qui indiqueront aux autres si un chemin est intéressant.
- Les chemins fortement phéromonés indiquent les chemins vers de bonnes solutions, car traversés à une fréquence plus importante.
- Exploration : à chaque choix (embranchement) une fourmi effectue un choix aléatoire, probabilités proportionnelle au taux de phéromone.
- Mécanismes de diversification pour diminuer le risque d'enfoncer la colonie dans un minimum local : évaporation des phéromones, pour rehausser l'intérêt des autres chemins. Le taux d'évaporation de phéromones, mais aussi bornes minimales et maximales.

ACO, illustration



Path relinking : un nouvel opérateur

- Path relinking : opérateur de construction de nouvelles solutions
- A partir de deux solutions réalisables, on trace un chemin entre ces deux solutions (ex : avec distance de Hamming de 1 pour chaque déplacement)
- On évalue les solutions définies par le chemin, à la recherche d'améliorations.
- Path relinking utilisé dans Scatter Search, mais aussi avec une heuristique GRASP

Scatter Search, des références

F. Glover, "Heuristics for integer programming using surrogate constraints", Decision Sciences, 1977.

F. Glover, "Tabu Search for Nonlinear and Parametric Optimization (with Links to Genetic Algorithms)", Discrete Applied Mathematics, 1994.

F. Glover, "A Template For Scatter Search And Path Relinking", in Artificial Evolution, pages 13, Sprinter, 1998.

F. Glover, "Scatter search and path relinking", in New Ideas in Optimization, pages 297–316, McGraw-Hill Ltd., 1999.

F. Glover and M. Laguna and R. Martí, "Fundamentals of Scatter Search and Path Relinking", Control and Cybernetics, 2000.

F. Glover and M. Laguna and R. Martí, "Scatter Search", in Advances in Evolutionary Computation : Theory and Applications, pages 519–537, Springer-Verlag, 2003.

M. Laguna and R. Martí, "Scatter search : methodology and implementations in C", Kluwer Academic Publishers, 2003.

R. Martí and M. Laguna and F. Glover, "Principles of Scatter Search", European Journal of Operational Research, 2006.

Quelques mots sur la parallélisation

- Les schémas de parallélisations évoqués pour les méta-heuristiques de trajectoires restent applicables avec les méta-heuristiques de population. De plus :
- Séparer la population en sous-population permet des implémentations distribuées, où des échanges (migrations) de solutions permettent d'apporter plus de diversification. (ex : islands models)
- La recherche peut être coordonnée par un noeud indiquant des zones de recherche diversifiées à d'autres noeuds, avec des schémas de parallélisation maître-esclaves.
- Dans le cas multi-objectif, la diversification des recherches peut se faire sur différentes zones d'un front de Pareto

Nedjah N., Alba E and L. de Macedo Mourelle. Parallel evolutionary computations. Springer, 2006. (freely available online)

Luna, F., Alba, E., Nebro, A. J. (2005). Parallel Heterogeneous Metaheuristics. Parallel Metaheuristics : A new Class of Algorithms, 47, 395.

Alba, E., Luque, G., Nesmachnow, S. (2013). Parallel metaheuristics : recent advances and new trends. International Transactions in Operational Research, 20(1), 1-48.

Algo évolutionnaire et parallélisation

- mutation = recherche locale, on peut reprendre des itérations Hill Climbing. On peut utiliser une parallélisation OpenMP
- Parallélisation MPI haut-niveau : On peut gérer des populations différentes dans chacun des processus.
- Opérations de croisement/mutations sur une population localement à un noeud sont des opérations indépendantes avec mémoire partagée : OpenMP
- Communication pour récupérer de bonnes solutions vers d'autres noeuds, parallélisation MPI : intérêt, évite des phénomènes de "consanguinité".
- On peut aussi garder un grand sous-ensemble de noeuds pour un algo évolutionnaire, et garder quelques processus pour faire tourner des itérations GRASP (ou autre recherche locale) : intérêt de grouper les processus et de définir un communicateur spécifique pour les processus impliqués dans l'algo évolutionnaire.
- Remarque : intérêt de parallélisation MPI encore plus marquant pour extension multi-objectif sans préférence, génération de grands fronts de Pareto (cf cours de M2).

Plan

Taxonomie

- Méta-heuristiques constructives/perturbatives : construit une solution ex nihilo/ améliore une solution existante par transformations locales
- Méta-heuristiques de trajectoire/populations : se base sur une solution courante unique/sur une population de solutions
- Méta-heuristiques inspirées de phénomènes naturels : recuit simulé, colonies de fourmis, algorithmes évolutionnaires
- Méta-heuristiques hybrides.
- Points communs clés : encodage, voisinages, intensification, diversification, mémoire.

Intensification/Diversification

- Diversification : mécanismes pour une exploration assez large de l'espace de recherche.
- Intensification : exploitation de l'information accumulée durant la recherche et concentration sur une zone précise prometteuse.
- Importance de bien calibrer ces mécanismes antagonistes
- Trop d'Intensification : risque de rester dans même zone, et de rester dans un minimum local.
- Aucune intensification : marche aléatoire, exploration de régions peu prometteuses.

Meta-heuristiques et mémoires

- La recherche tabou implémentait un mécanisme de mémoire.
- La mémoire peut aider à calibrer le compromis diversification / intensification.
- Des mécanismes de mémoires peuvent être greffés sur les méta-heuristiques précédemment évoquées.
- Ex, en ayant plusieurs voisinages, comment choisir un voisinage donné lorsqu'il y a plusieurs possibilités ?
- Taux d'acceptation : sur les N dernières itérations où un voisinage a été utilisé, à quelle fréquence y a t'il eu des améliorations ?
- Autre critère : écarts d'améliorations induit par un voisinage divisé par le temps d'exploration du voisinage (pour exploration de voisinage entier, comme Hill Climbing-VND)
- Lorsque plusieurs choix de voisinage (ou d'opérateurs) sont possibles, les probabilités de choix peuvent être adaptées suivant un critère tel que le taux d'acceptation.

No Free Lunch Theorem

- No Free Lunch Theorem : aucune méta-heuristique ne surpasse les autres sur tous les problèmes d'optimisation.
- Seule une comparaison empirique sur un problème et des instances données permet de comparer des approches.
- Evaluation d'une méta-heuristique non-déterministe : 25-30 runs suivant des réalisations aléatoires données, la qualité de la meilleure solution importe moins que la dispersion des solutions (par exemple analyser avec des quartiles)
- Sur un problème donné, une méta-heuristique peut s'avérer plus adaptée qu'une autre.
- Point clé : chiffrer l'impact d'opérateurs, valider les varaintes d'opérateurs et leur probabilité d'utilisation par exemple
- N.B : la définition d'opérateurs, tels que les voisinages a un grand impact sur les performances finales des meta-heuristiques.

Heuristiques hybrides

- Opérateurs de diversification/exploration des méta-heuristiques assez interchangeables
- Grand potentiel d'hybridation entre méta-heuristiques
- ex : recuit simulé comme opérateur de mutation d'un algorithme génétique
- Hybridation naturelle avec algos exacts spécifiques (Prog dynamique), PPC, pour l'exploration d'un voisinage avec respect de contraintes.
- Matheuristiques : hybridation Programmation mathématique et méta-heuristiques. ex :
  - exploration de grands voisinages, gestion de contraintes dans des voisinages.
  - construction de solution guidée par la relaxation continue

References

• T. Adamo, G. Ghiani, A. Grieco, E. Guerriero, and E. Manni. *MIP neighborhood synthesis through semantic feature extraction and automatic algorithm configuration*. Computers & Operations Research, 83 :106–119, 2017.

• E-G. Talbi. *Combining metaheuristics with mathematical programming, constraint programming and machine learning*. Annals of Operations Research, 240(1) :171–215, 2016.

• C. Blum, J. Puchinger, G. Raidl, and A. Roli. *Hybrid metaheuristics in combinatorial optimization : A survey*. Applied Soft Computing, 11(6) :4135–4151, 2011.

• L. Jourdan and M. Basseur and E.-G. Talbi, *Hybridizing exact methods and metaheuristics : A taxonomy*, European Journal of Operational Research, 199 (3) :620-629, 2009

• Boschetti, M. A., Maniezzo, V., Roffilli, M., Röhler, A. B. . *Matheuristics : Optimization, simulation and control*. In International Workshop on Hybrid Metaheuristics (pp. 171-177). 2009.

• G. Raidl and J. Puchinger. *Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization*. Computational Intelligence, 114 :31–62, 2008.

Méta-heuristiques et contraintes

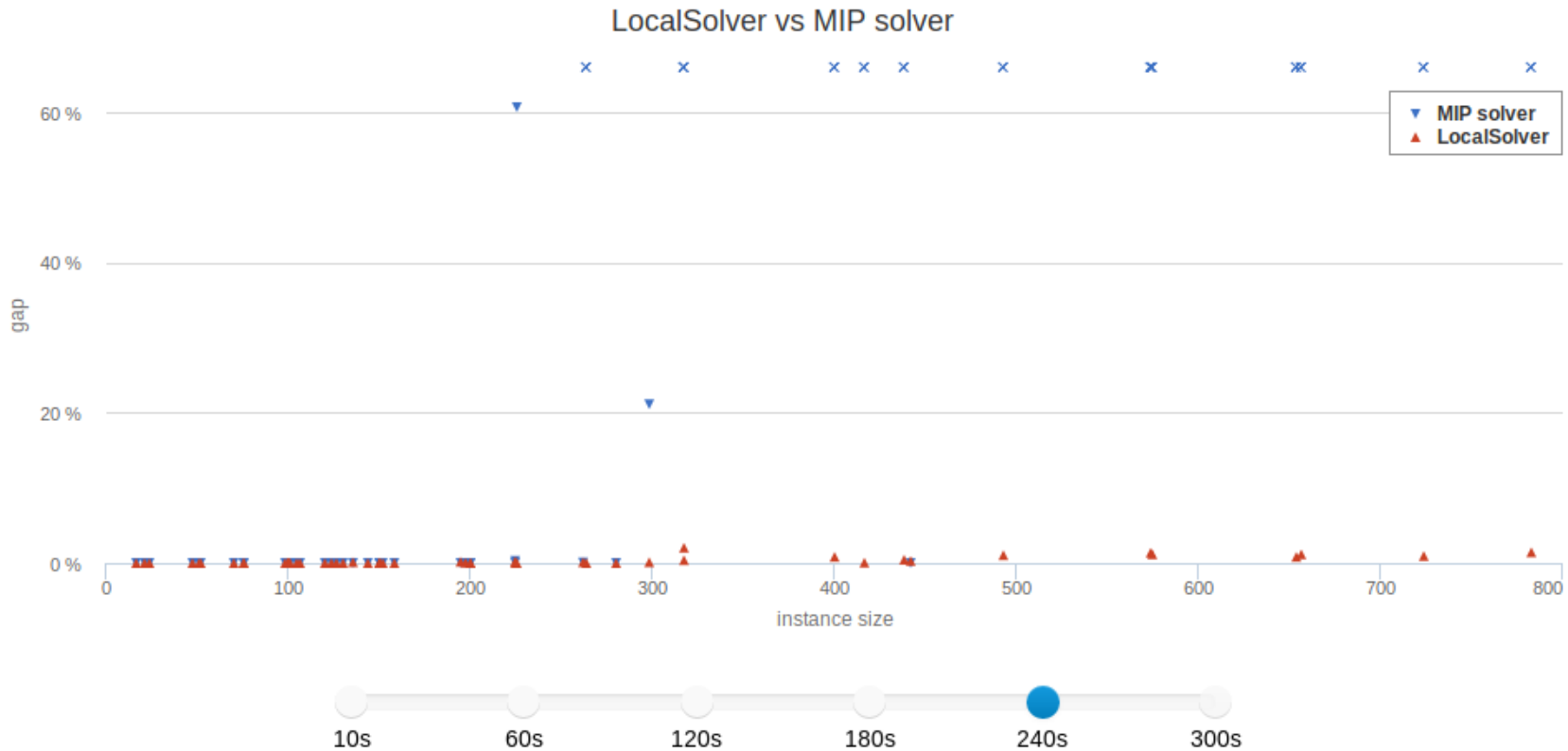
- Les problèmes fortement contraints posent des difficultés aux méta-heuristiques pures.
- Ex : une solution construite, et pas de solution réalisable dans le voisinage local ....
- Lorsque l'encodage comprend les contraintes, on a la garantie de ne pas être dans de tels cas.
- Une pratique : pénalisation de contraintes, mettre les contraintes en fonction objectif avec un fort poids, de sorte à favoriser les solutions réalisables.
- Difficulté : si on n'a jamais de solution réalisable au cours de la recherche, on ne peut rien conclure sur l'existence de solution. Une méthode exacte peut prouver la non existence de solution.
- De manière générale, utiliser de la PPC ou une approche exacte (PLNE, programmation dynamique) avec des grands voisinages qui satisfont les contraintes permet de mieux gérer des problèmes fortement contraints

Avantages des solveurs PLNE

- Bcp de problèmes industriels se modélisent en PLNE. Solveurs PLNE permettent de résoudre génériquement des PLNE en "model&run" avec l'algorithme de Branch&Bound (B&B).
- Des solveurs commerciaux efficaces avec licence académique : Cplex, Gurobi, Xpress , SCIP . . .Cplex disponible à l'université
- Des solveurs open source : COIN-OR (cbc, clp), GLPK.
- Les PL se résolvent en temps polynomial (mais l'efficacité pratique varie sensiblement selon les solveurs).
- Mode exact : terminaison avec une solution optimale (avec une tolérance définie, par défaut avec Cplex : 0.01%)
- En temps contraint : fourni la meilleure solution trouvée, et des bornes inférieures et supérieures pour encadrer la valeur optimale.

MAIS ?

Inconvénients des solveurs PLNE



- Solveur PLNE converge en temps exponentiel en général, lié aux caractéristiques de la recherche arborescente.
- En temps contraint, les solveurs PLNE deviennent inefficaces lorsque la taille des PLNE augmente
- En contexte d'application industrielle : taille des instances et temps de résolution sont souvent imposés.

Méta-heuristiques : avantages et inconvénients

Avantages des méta-heuristiques :

- Explorent bien plus de solutions en en temps contraint qu'une recherche arborescente B&B.
- Passent mieux à l'échelle : les méta-heuristiques sont efficaces pour trouver de bonnes solution for des temps courts de résolution sur des grandes instances, contrairement aux solveurs PLNE.
- Les (méta-)heuristiques nécessitent peu de puissance de calcul, permettent du calcul embarqué (robotiques).
- La parallélisation sur GPGPU est compatible et utilisable pour des méta-heuristiques, ce n'est pas le cas pour la recherche arborescente B&B.

Inconvénients des méta-heuristiques :

- Convergent et trouvent uniquement des optimums locaux, pas forcément globaux.
- Pas de garantie d'optimalité, pas de borne duale pour garantir la qualité de la meilleure solution trouvée.
- Difficultés surviennent pour des instances fortement contraintes (ensemble de solutions réalisable épars).

$\Longrightarrow$ La résolution exacte PLNE et les heuristiques ont des avantages et inconvénients complémentaires.

$\Longrightarrow$ Matheuristiques : hybridons programmation mathématique et heuristiques !

Accélérer la recherche arborescente B&B exacte

- Pour l'efficacité de la recherche arborescente B&B exacte, il est crucial d'avoir le PLNE le plus simple à résoudre
- Techniques de preprocessing pour fixer a priori des variables à leur valeur optimale.
- Reformulations et agrégations permettent de résoudre un PLNE équivalent et plus petit.
- Jusqu'aux années 90, un effort principalement à améliorer la résolution B&B avec des coupes d'intégrité.
- Critère d'élagage dans l'arbre B&B : borne duale du noeud $>$ meilleure solution trouvée (BKS) - tolérance
- Améliorer les coupes (techniques de maths) permettent d'élaguer plus tôt certains noeuds et d'énumérer moins de noeud au final.

Savelsbergh, M.W. : Preprocessing and probing techniques for mixed integer programming problems. ORSA Journal on Computing 6(4), 445–454 (1994)

Bixby, E. R., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R. . MIP : Theory and practice–closing the gap. In IFIP Conference on System Modeling and Optimization (pp. 19-49). Springer, 1999.

Heuristiques à l'intérieur des solveurs PLNE

- Avoir au plus tôt de bonnes solutions (et idéalement l'optimum)a également un effet positif sur l'activation de l'élagage.
- Depuis les années 2000, des efforts à améliorer la recherche de solutions primales, accélère la résolution exacte à l'optimalité, et améliore la recherche en temps limité.
- Heuristiques constructive : stratégies de plongements (diving) pour chercher des solutions réalisables, Feasibility Pump, RENS
- Heuristiques perturbatives, ie améliorant une solution existante connue : RINS, Local Branching, VNS-Local Branching.
- "Polishing", polir la meilleure solution trouvée à al fin de la recherche B&B pour essayer de l'améliorer.

$\Longrightarrow$ Actually, MILP solvers can be seen as matheuristics in defined time limits.

Références

• Achterberg, T., Koch, T., Martin, A. : Branching rules revisited. Operations Research Letters 33(1), 42–54 (2005)

• Berthold, T. RENS. Mathematical Programming Computation, 6(1), 33-54. (2014).

• Berthold, T. : *Primal Heuristics for Mixed Integer Programs*. Master thesis, Berlin, (2006).

• L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. Discrete Optimization, 4(1) :63–76, 2007.

• E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. Mathematical Programming, 102 :71–90, 2005.

• Hansen, P., Mladenović, N., Urošević, D. Variable neighborhood search and local branching. Computers & Operations Research, 33(10), 3034-3045. (2006).

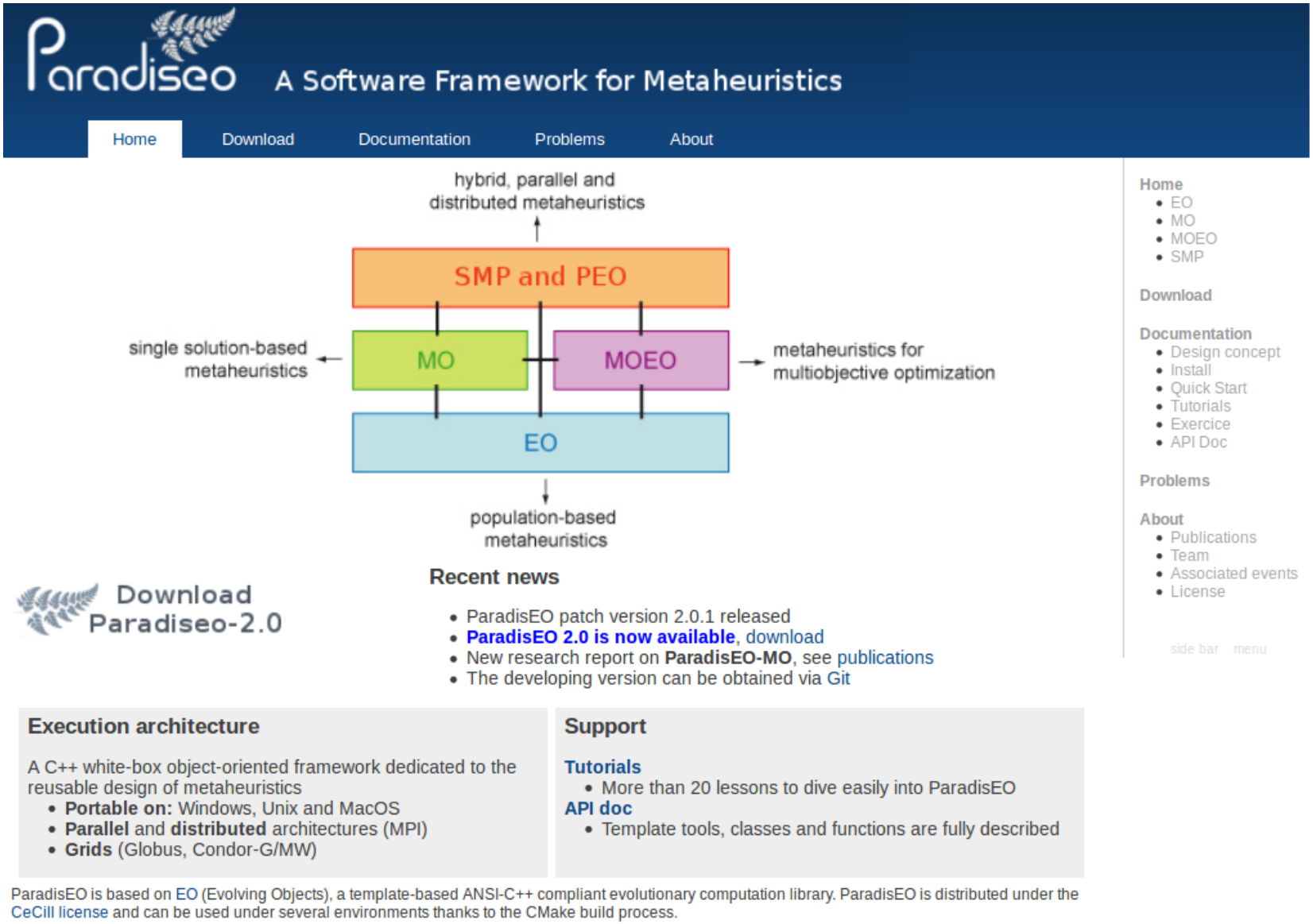
• M. Fischetti and A. Lodi. Local branching. Mathematical Programming, 98(1-3) :23–47, 2003.

• Rothberg, E. : *An evolutionary algorithm for polishing mixed integer programming solutions* INFORMS Journal on Computing, vol 19, n 4, pp 534–541, 2007.

Plan

Outils implémentant des algorithmes évolutionnaires

- Plusieurs outils/bibliothèques codent des algos évolutionnaires génériques.
- Opérateurs génériques basés sur aléatoire.
- Généricité au dépend de qualité utilisation de voisinages spécifiques : particulièrement prégnant sur des problèmes fortement contraints.
- De telles bibliothèques peuvent s'utiliser sur des problèmes peu contraints (idéalement sans contrainte)
- Utilisation en ingénierie dans un cadre multi-objectif.



http ://paradiseo.gforge.inria.fr/index.php ?n=Main.HomePage

Fonctionnalités Paradiseo

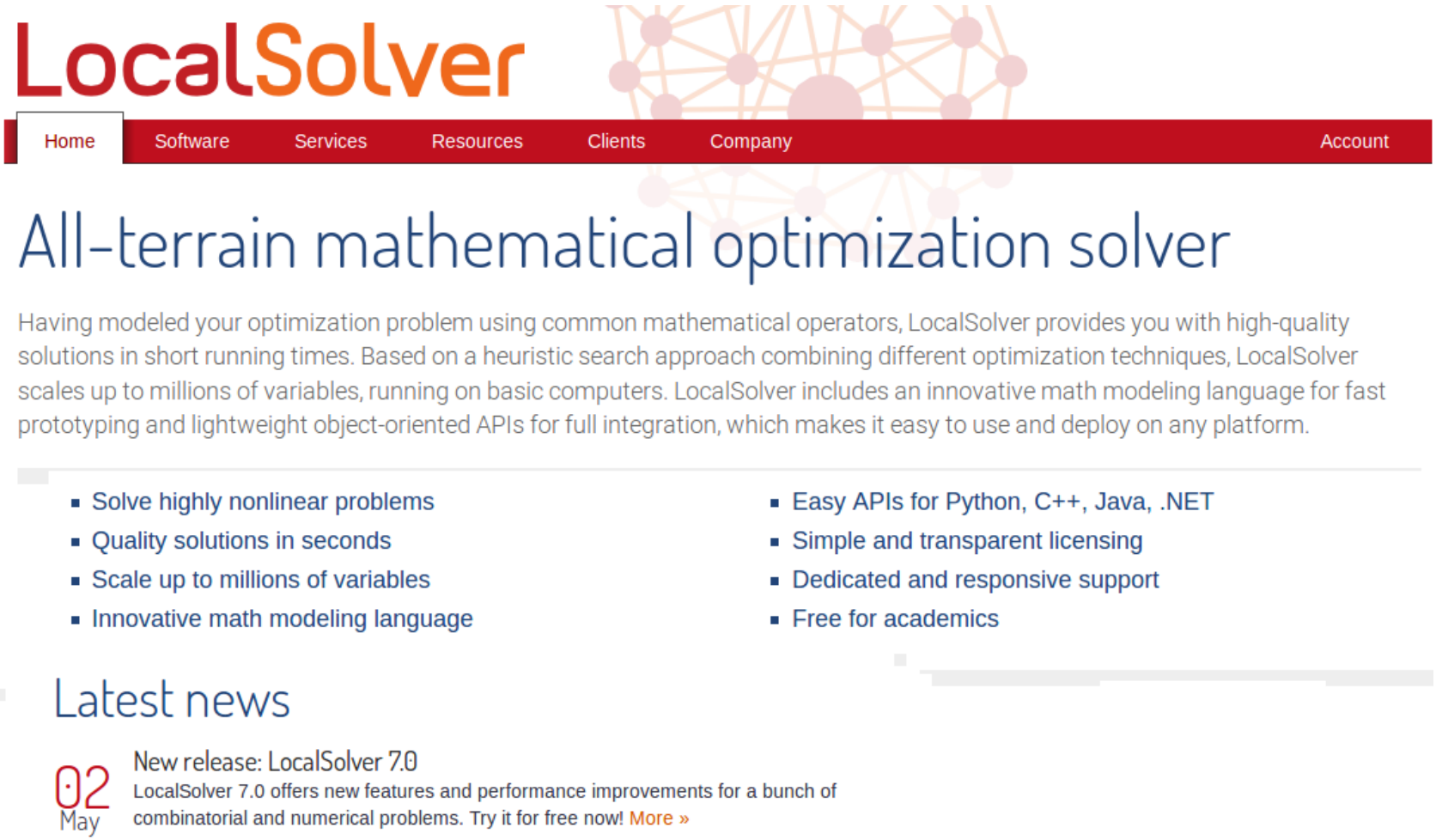
- Solveur "boîte blanche" : l'utilisateur peut définir ses voisinages et opérateurs de méta-heuristiques
- Idéal pour expérimenter et croiser plusieurs opérateurs de méta-heuristiques
- Implémentation parallèle de qualité.
- Fonctionnalités d'analyse : paysages, statistiques...
- Vision unifiée des méta-heuristiques, interchangeabilité d'opérateurs de méta-heuristiques

Cahon, S., Melab, N., Talbi, E.G. : Paradiseo : A framework for the reusable design of parallel and distributed metaheuristics. Journal of Heuristics 10 (3), 357–380 (2004)

Talbi, E.G. : Metaheuristics : from design to implementation, John Wiley & Sons, 2009

Talbi, E.-G. : A Taxonomy of Hybrid Metaheuristics, Journal of Heuristics, 8, 2002, 541-564

LocalSolver



http ://www.localsolver.com/

Fonctionnalités LocalSolver

- Fonctionnalités model&run avec un langage de modélisation naturel.
- Algorithme de recherche locale en boîte noire, enrichi avec des aspects/structures PPC et PLNE suivant le type de problème.
- Non basé sur une recherche arborescente exponentielle : permet de mieux passer à l'échelle en grande taille.
- Opérateurs de méta-heuristiques codés suivant des structures spécifiques : le choix d'écriture de modélisation n'est pas neutre, ne pas modéliser avec LocalSolver comme en PLNE.
- On peut définir une variable d'un problème d'optimisation comme une liste d'éléments distincts ou une partition (ex : TSP).
- Langage de modélisation spécifique, et API C/C++, Python, Java.
- TSP et p-median dans les exemples simples de LocalSolver

https ://www.localsolver.com/documentation/exampletour/pmedian.html

LocalSolver et TSP

```
/* Declares the optimization model. */
function model() {
    // A list variable: cities[i] is the index of the ith city in the tour
    cities <- list(nbCities);

    // All cities must be visited
    constraint count(cities) == nbCities;

    // Minimize the total distance
    obj <- sum(1..nbCities-1, i => distanceWeight[cities[i-1]][cities[i]])
            + distanceWeight[cities[nbCities-1]][cities[0]];

    minimize obj;
}

/* Parameterizes the solver. */
function param() {
    if (lsTimeLimit == nil) lsTimeLimit = 5;
}
```

LocalSolver et p-median

```
/* Declares the optimization model. */
function model(){
    // One variable for each location
    x[1..N] <- bool();

    // No more than p locations are selected
    constraint sum[i in 1..N] (x[i]) <= p;

    // Costs between location i and j is w[i][j] if j is selected in S or 2*wmax if not
    costs[i in 1..N][j in 1..N] <- x[j] ? w[i][j] : 2*wmax;

    // Cost between location i and the closest selected location
    cost[i in 1..N] <- min[j in 1..N] (costs[i][j]);

    // Minimize the total cost
    totalCost <- sum[i in 1..N] (cost[i]);
    minimize totalCost;
}

/* Parameterizes the solver. */
function param(){
    if(lsTimeLimit == nil) lsTimeLimit = 10;
}
```

Oscar

- Intermédiaire 'entre "black-box" LocalSolver et "white-box" Paradiseo
- Résolution par recherche locale et un solveur de PPC.
- Des voisinages définis et possibilité de définir ses voisinages
- Un formalisme de modélisation

`https://oscarlib.readthedocs.io/en/latest/`

`https://bitbucket.org/oscarlib/oscar/wiki/Home`

Plan

Ce qu'il faut retenir des méta-heuristiques

- Optimisation locale : voisinages, exploration rapide de bonnes solutions.
- Optimisation locale : on trouve des minimums locaux, pas de garantie de trouver des minimums globaux en général.
- Famille de méthodes heuristiques à spécifier sur un problème donné
- Points communs clés : encodage, voisinages, intensification, diversification, mémoire. (N.B : des similitudes en ML où le concept intensification/diversification est également décrit)
- En pratique, les méta-heuristiques sont moins limitées par la taille des problèmes que les approches d'optimisation exactes/globales.
- En pratique, des problèmes fortement contraints peuvent être très handicapants pour des approches d'optimisation locale.

Perspectives

- Des techniques à mettre en pratique, l'expertise s'acquiert avec l'expérience, en se frottant à différents types de problèmes d'optimisation.
- De nombreux liens, de nombreuses variantes à approfondir. Les principes généraux et la vision d'ensemble donnée par ce cours est un bon guide pour approfondir.
- Méta-heuristiques (et parallélisation) adaptées pour une extension en optimisation multi-objectif. Cours de M2 !