IA une solution : la recherche locale stochastique.

Capsule pédagogique d'initiation à la conception d'applications de résolution de problèmes de décision et d'optimisation combinatoire à l'aide de la recherche locale stochastique.

Olivier Bailleux. Version du 27 juillet 2020.

Objectif

Être capable de concevoir une application de **recherche locale stochastique** permettant de résoudre un problème de décision ou d'optimisation combinatoire (beaucoup) plus rapidement que par une recherche d'une solution au hasard.

Prérequis

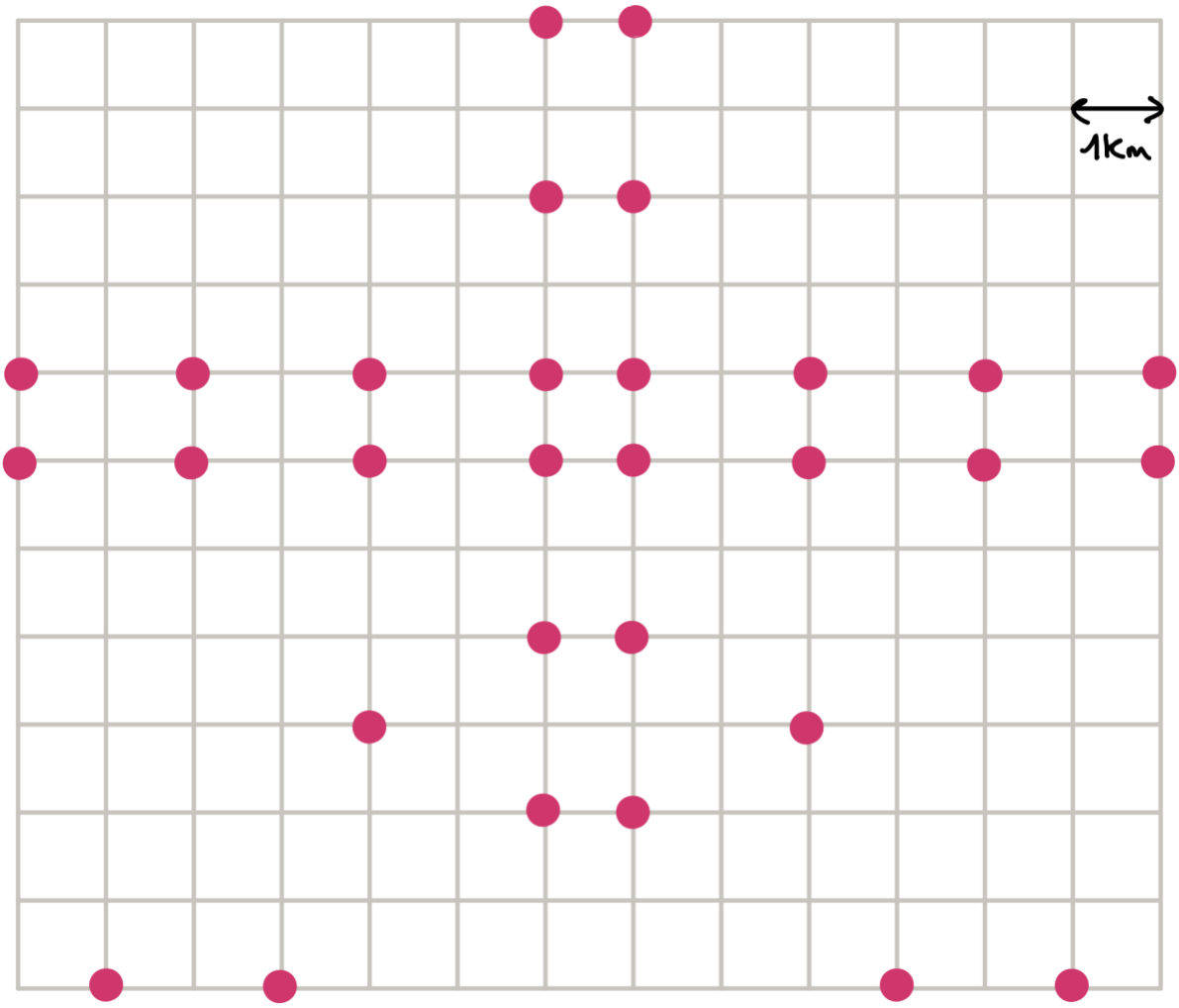
Notions élémentaires d'algorithmique (variable, boucle, alternative). Comprendre ce qu'est une probabilité, un graphe, une fonction, un extremum d'une fonction, la fonction exponentielle. Comprendre les notations mathématiques faisant intervenir des fonctions et des opérations arithmétiques de bases (produit, fraction, puissance, différence, somme). Une première expérience de programmation impérative (Java, Python, C, C++…).

Introduction

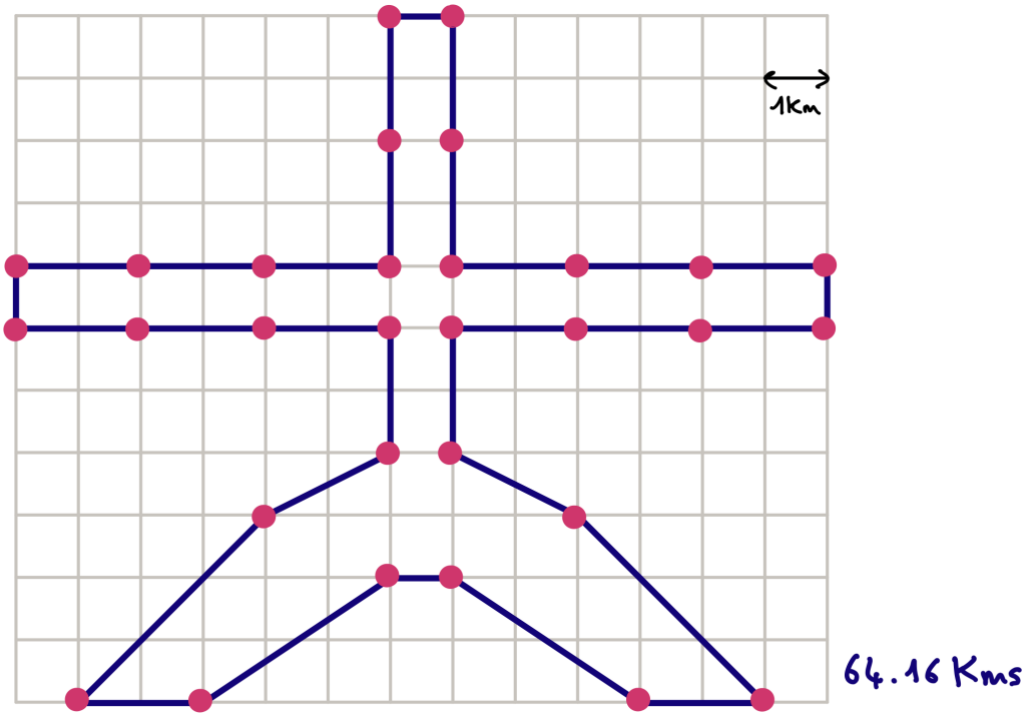
Aujourd'hui je vais vous présenter une méthode permettant de résoudre des problèmes à l'aide d'un ordinateur : la recherche locale stochastique. Le nom est un peu effrayant mais le principe est plutôt simple et facile à programmer, même pour des débutants.

Je vais présenter cette technique à l'aide d'un exemple appelé le **problème du voyageur de commerce** :

Comment produire un circuit de longueur minimale passant par chacune de ces 30 villes ?



Voici ma solution.



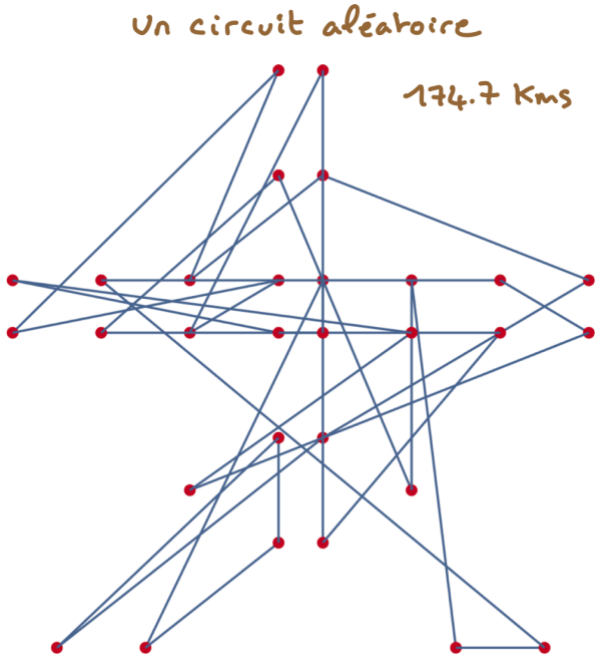
Mais comment un ordinateur peut-il traiter efficacement ce problème ? Pas seulement ce cas particulier, qui est en fait une **instance** du problème, mais n'importe quelle instance, quel que soit le nombre de villes et leurs dispositions ?

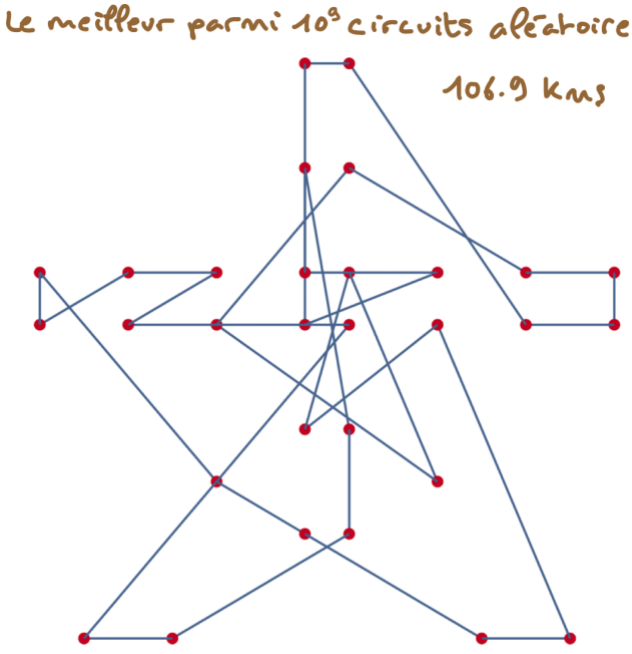
Derrière son apparence anodine, ce problème est difficile à résoudre. D'ailleurs la solution présentée ci-dessus n'est pas optimale. Le circuit fait 64.16 kms alors qu'il en existe qui font 60.26 Kms. Je n'ai pas réussi à en trouver un sans ordinateur. Vous pouvez essayer, mais ne perdons pas de vue que le but est de résoudre ce problème à l'aide d'un ordinateur et non à la main.

Le hasard pur

Une approche extrêmement simple consiste à produire des circuits au hasard, calculer leurs longueurs et sélectionner le plus court.

Voici un circuit aléatoire et le meilleur circuit obtenu en produisant un milliard de circuits aléatoires. Cela a pris 10 minutes à un petit programme JavaScript sur mon ordinateur de bureau. Sa longueur est 106.9 kms.





Plusieurs remarques :

1. Lorsque j'ai commencé à programmer des ordinateurs dans les années 1980, produire un milliard de circuits et calculer leurs longueurs aurait pris plusieurs semaines, ce qui montre bien l'évolution de la puissance des machines.

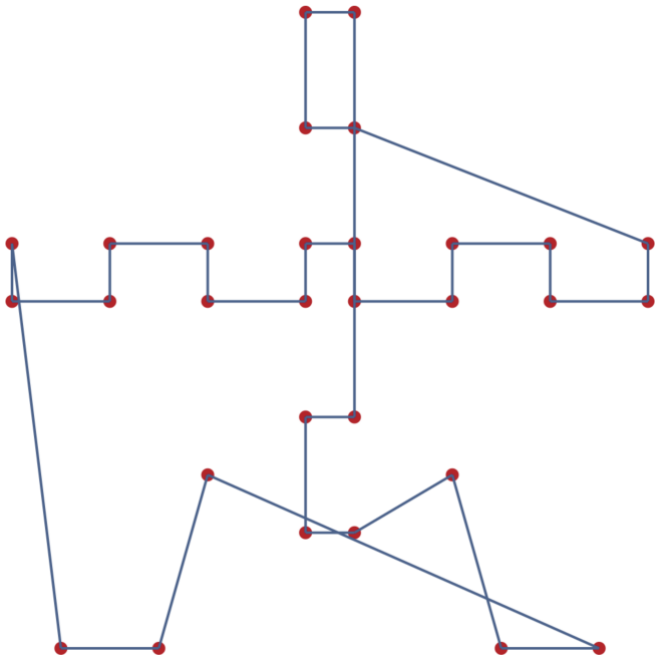
2. Le résultat obtenu est malgré tout très mauvais : plus 106 Kms alors que le circuit optimal en fait 60.3. Et même si j'avais produit des milliards de milliards de circuits au hasard, j'aurais eu très peu de chance d'obtenir une solution optimale, parce qu'il y a plus de 4000 milliards de milliards de milliards de circuits différents dont très peut sont de longueur optimale.
3. Donc pour résoudre des problèmes difficiles, la puissance de calcul ne suffit pas. La force brute ne suffit pas. Il faut des **algorithmes astucieux et efficaces**. Et c'est une des raisons pour lesquelles il y a une recherche très active en science informatique.

Une heuristique simple

Plutôt que produire des circuits au hasard, on peut utiliser une **heuristique**, c'est-à-dire un algorithme donnant souvent des résultats acceptables mais sans aucune garantie d'optimalité.

L'heuristique que je propose consiste à partir d'une ville, puis à rallier la ville la plus proche, puis la ville pas encore visitée la plus proche de la dernière ville visitée, et ainsi de suite.

On obtient alors le circuit suivant qui a une longueur de 70.5 kms. C'est beaucoup mieux qu'avec l'approche aléatoire, et beaucoup plus rapide, mais on est toujours loin de la longueur optimale de 60.3 kms.



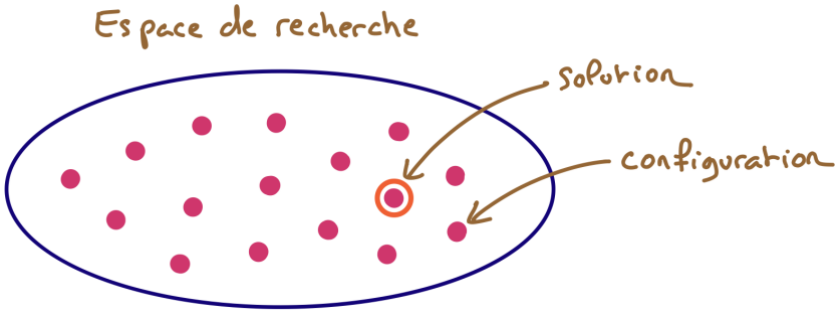
Le hasard apprivoisé

Le principe et les ingrédients de la recherche locale stochastique

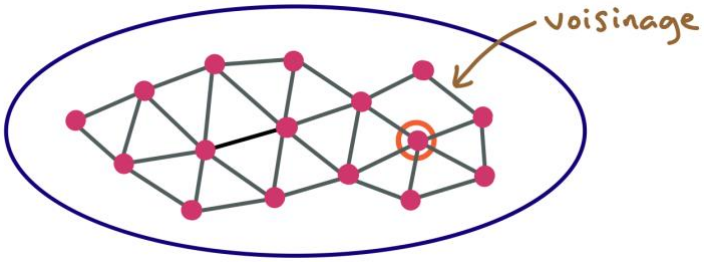
Nous allons maintenant découvrir comment trouver un circuit de très bonne qualité, et même optimal, à l'aide de la recherche locale stochastique. L'idée consiste à partir d'un circuit produit aléatoirement, puis à l'améliorer par petites transformations successives, en laissant une place au hasard pour des raisons qui seront précisées plus tard.

Les ingrédients pour utiliser la recherche locale stochastique sont les suivants :

1. Un **espace de recherche** qui est un ensemble dont les éléments sont appelés **configurations**. Certaines de ces configurations doivent représenter les solutions recherchées.

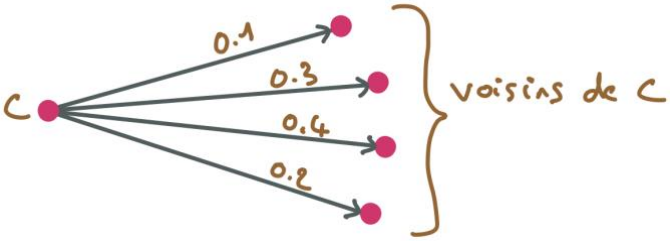


2. Un **voisinage**, qui associe à toute configuration de l'espace de recherche des configurations dites voisines. L'idée est que deux configurations voisines doivent être ressemblantes. Le passage d'une configuration à une de ses voisines traduit l'idée de « petite modification ».



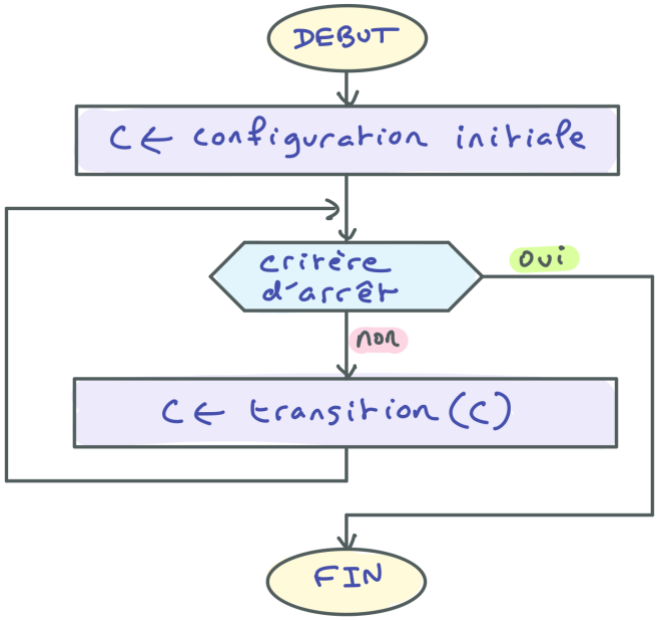
3. Un **critère d'initialisation** qui détermine la configuration à partir de laquelle on va commencer la recherche.

4. Un **critère de transition** qui permet, étant donnée une configuration C, de choisir une de ses voisines. Ce critère de transition doit faire intervenir le hasard, mais en privilégiant certains voisins par rapport aux autres. Il assigne donc une probabilité à chaque voisine de C.



5. Un **critère d'arrêt** qui permet de décider quand on arrête la recherche.

Le schéma de l'algorithme est le suivant.



On part d'une configuration choisie selon le critère d'initialisation, appelée **configuration courante**. Tant que le critère d'arrêt n'est pas réalisé, on remplace la configuration courante par une de ses voisines, choisie par le critère de transition. Le résultat est la dernière configuration courante obtenue.

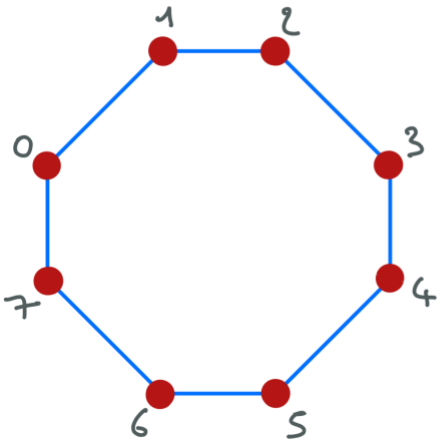
Les ingrédients pour le voyageur de commerce

Examinons les ingrédients que j'ai spécialement sélectionné pour le problème du voyageur de commerce.

L'espace de recherche

Les N villes sont numérotées de 0 à $N-1$ et chaque circuit est représenté par une permutation de ces entiers, c'est-à-dire une liste dans laquelle chaque entier apparaît exactement une fois. Cette liste indique l'ordre de visite des villes quand on suit le circuit.

Un circuit



Deux représentations de ce circuit

1	2	3	4	5	6	7	0

5	6	7	0	1	2	3	4

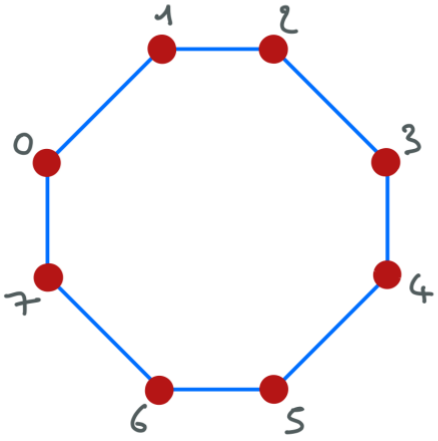
Il existe $2N$ représentations différentes d'un même circuit reliant N villes. C'est un choix. On pourrait imposer par exemple que la première valeur de la liste soit 0. Il y aurait alors N fois moins de configurations, mais la densité des configurations représentant les solutions serait la même et cela n'aurait aucun intérêt pratique pour la suite.

On pourrait utiliser d'autres espaces de recherche, mais celui-ci me paraît être le plus naturel pour ce problème.

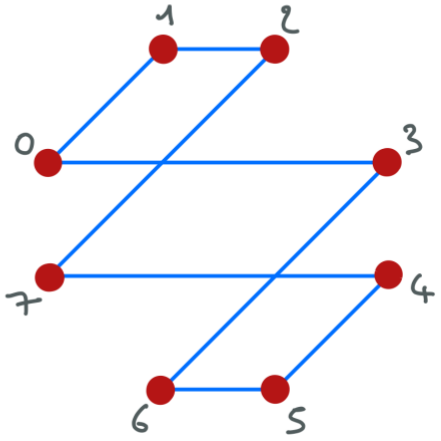
Le voisinage

Je propose deux exemples de voisinages. Avec le premier, que j'appellerai voisinage A, on passe d'une configuration à une de ses voisine en échangeant deux villes.

1	2	3	4	5	6	7	0

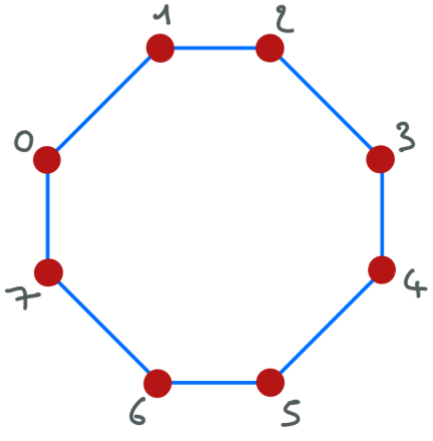


1	2	7	4	5	6	3	0

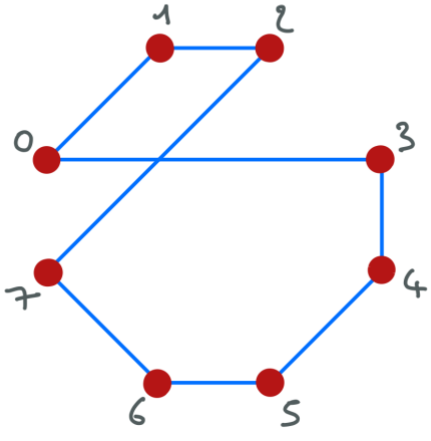


Avec le second, le voisinage B, on passe d'une configuration à une de ses voisines en inversant une suite de plusieurs villes.

1	2	3	4	5	6	7	0



1	2	7	6	5	4	3	0



Si on observe les configurations, on a l'impression que le voisinage B entraîne un changement plus important que le voisinage A, mais si on observe les circuits, on a plutôt l'impression inverse. C'est le voisinage A qui entraîne le plus grand changement.

A votre avis, lequel des deux voisinages sera le plus efficace ? Les pronostics sont ouverts.

Le critère d'initialisation

Comme c'est le cas très souvent, nous allons initialiser la recherche avec une configuration produite de manière aléatoire.

Le critère de transition

Bien que ce ne soit pas une obligation, le critère de transition peut être basé sur une **fonction coût** qui est minimale pour les solutions du problème, ou pour ses solutions optimales s'il s'agit d'un problème d'optimisation. Dans le cas du voyageur de commerce, la fonction coût la plus évidente consiste à associer à chaque configuration la longueur du circuit correspondant.

Un bon critère doit favoriser les transitions qui améliorent le coût, mais sans rejeter toutes les autres, de manière à éviter un blocage de la recherche, comme nous le verrons lors des expérimentations présentées dans la suite.

Nous allons utiliser le **critère de Metropolis**. Son principe est le suivant : On tire au hasard une configuration voisine V de la configuration courante C. Si le coût de V est inférieur ou égal à celui de C, on accepte V comme nouvelle configuration courante. Sinon on accepte V avec une probabilité calculée à l'aide de la formule suivante :

$$P = e^{\frac{f(C)-f(V)}{T}}$$

Dans laquelle f désigne la fonction coût et T est un paramètre appelé **température**. Lorsque la température est proche de 0, les transitions défavorables, celles qui augmentent le coût, sont presque toutes rejetées. Au contraire, lorsque la température est très élevée toutes les transitions sont acceptées et l'exploration de l'espace de recherche prend la forme d'une marche aléatoire ne tenant pas compte de la longueur des circuits. Les meilleurs résultats sont obtenus avec une température intermédiaire qui doit être calibrée expérimentalement.

Dans l'algorithme appelé **recuit simulé**, la température est d'abord élevée puis est progressivement réduite. Mais dans nos expérimentations, nous nous contenterons d'une température fixe.

Le critère d'arrêt

Nous arrêterons la recherche après un nombre donné de transitions et tentatives de transitions ou lorsque le coût tombe en dessous d'un seuil donné.

Passons à l'action

Résultats

Après la théorie, la pratique. J'ai implémenté l'algorithme basé sur les critères qui viennent d'être présentés. Voici les résultats obtenus. Les valeurs indiquées sont les temps moyens nécessaires pour trouver une solution optimale avec chacun des paramètres mentionnés.

	$T \sim 0$	T optimale
Voisinage A	10s	1.2s
Voisinage B	0,01s	0,004s

Les températures optimales ont été déterminées expérimentalement et sont sensiblement différentes : 0.5 pour le voisinage A et 0.3 pour le B.

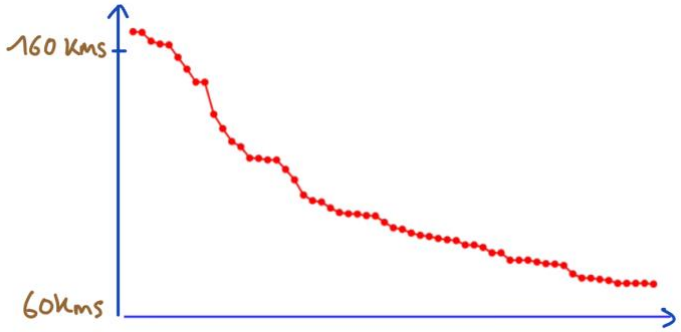
À la température optimale, le voisinage B est 300 fois plus performant que le voisinage A. À une température proche de 0, c'est-à-dire quand les transitions défavorables sont interdites, le voisinage B est 1000 fois plus performant que le A.

L'effet du réglage optimal de la température est nettement plus marqué avec le plus mauvais voisinage.

Observations expérimentales

Voisinage A à température proche de 0

Cette première séquence concerne le voisinage A avec une température proche de 0. Elle montre, pour chaque transition acceptée, le coût de la configuration courante. La ligne bleue représente le coût optimal qui est ici de 60,26 Kms.

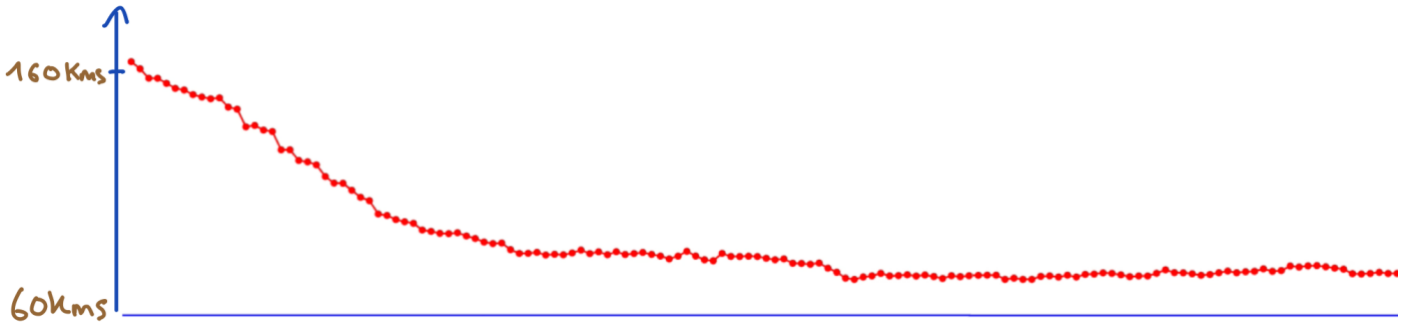


Dans un premier temps, le coût baisse progressivement, puis il n'y a plus aucune évolution (ou parfois on observe une oscillation entre configurations de même coût). La configuration courante est loin d'être optimale, mais elle ne peut plus être améliorée en une seule transition.

Pour trouver une solution optimale, l'algorithme va devoir faire de nombreux redémarrages à partir de nouvelles configurations initiales aléatoires.

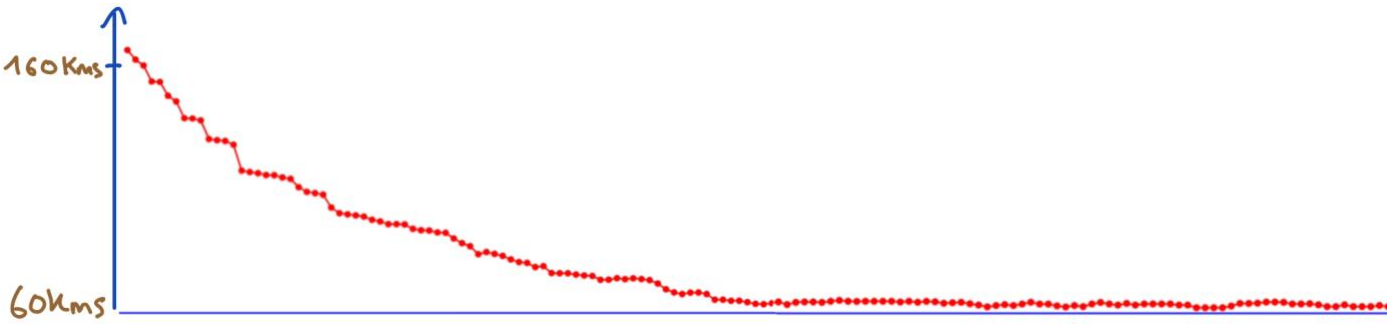
Voisinage A avec température optimale

Avec une température optimale, on a toujours une première phase d'amélioration progressive du coût, mais ensuite la recherche n'est pas bloquée. Autoriser des transitions défavorables permet à la recherche de s'extirper des extremums locaux de la fonction coût.



Il n'y a pas de convergence vers une solution optimale, mais plutôt l'exploration d'une partie de l'espace de recherche contenant des configurations de coûts nettement plus faibles que le coût moyen, incluant les solutions optimales. L'algorithme va finir par en trouver une, mais cela peut nécessiter de réaliser plusieurs transitions défavorables successives. Si on augmente la température, la recherche peut s'extraire plus facilement d'un ensemble de configurations où elle est piégée, mais on risquera alors de passer « à côté » d'une solution et de la manquer. Le réglage de la température est un compromis entre **l'intensification** de la recherche dans un ensemble de configuration à bas coût et sa **diversification** permettant d'éviter d'être piégé dans une partie de l'espace de recherche ne contenant aucune solution optimale.

Voisinage B avec température optimale



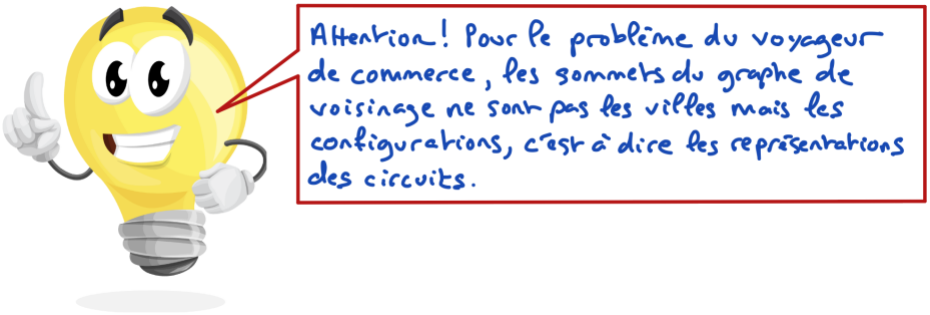
Le comportement de la recherche à température optimale avec le voisinage B est similaire à celui observé avec le voisinage A, mais après la phase d'amélioration progressive du coût, on constate que les configurations explorées ont des coûts nettement moins élevés. L'exploration est ciblée sur un ensemble de configurations nettement plus restreint que précédemment, ce qui permet de découvrir plus rapidement une solution optimale.

Bonnes pratiques de conception

Dans la suite, nous utiliserons le mot solution aussi bien pour désigner une solution optimale d'un problème d'optimisation telle que « trouver un circuit de longueur minimum » et pour désigner la solution d'un problème de décision tel que « trouver un circuit de longueur inférieure à une certaine valeur L ».

Paysage de recherche

Un voisinage définit un graphe dont les sommets sont les configurations de l'espace de recherche. L'association de ce graphe et de la fonction coût est appelée **paysage de recherche**.

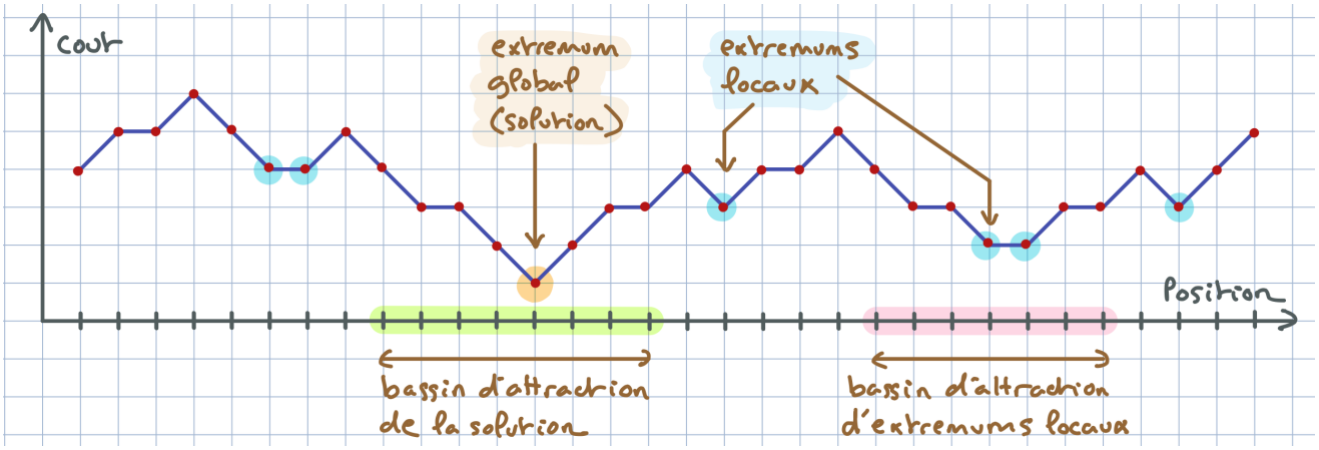


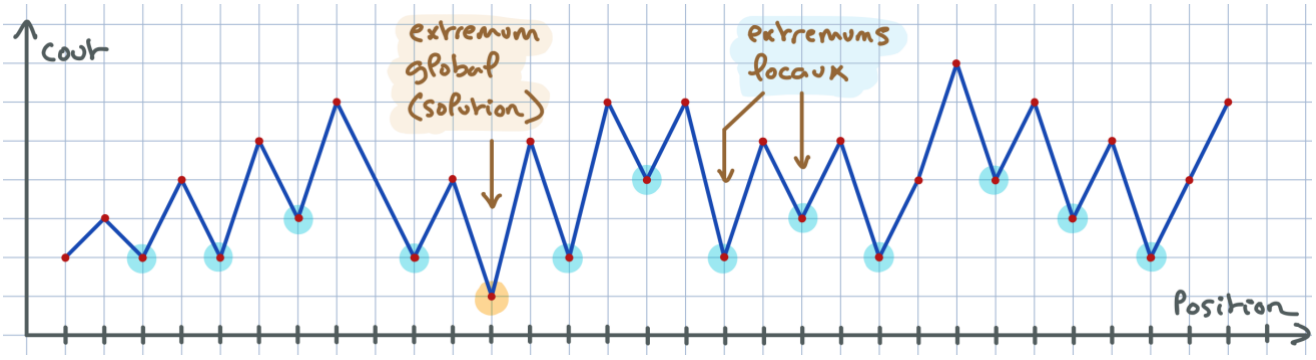
Les paysages de recherche des problèmes combinatoires « difficiles », comme ceux du problème de voyageur de commerce et des autres problèmes de satisfaction de contraintes ou d'optimisation sous contraintes, sont des objets mathématiques complexes que nous allons aborder ici de manière simplifiée et imagée.

Pourquoi le voisinage B donne-t-il de bien meilleurs résultats que le voisinage A ? C'est parce qu'avec le voisinage B le paysage de recherche est plus « lisse » alors qu'avec le voisinage A il est en quelques sorte plus « accidenté », plus « rugueux ».

Si chaque configuration n'avait que deux voisines et que le graphe de voisinage était connexe, alors il aurait une seule dimension à explorer, le coût pourrait être vu comme une altitude et l'objectif de la recherche serait de trouver le point le plus bas.

Voici deux exemples de tels paysages à deux dimensions, dont une représente le coût. Le voisinage utilisé autorise un déplacement d'une unité à gauche ou à droite.





Imaginez que vous deviez être parachuté dans un de ces paysages recouverts d'un épais brouillard. Vous n'avez aucune vision globale. Votre vision est limitée aux deux plus proches positions. Votre mission est de rechercher le point le plus bas, où se trouve un trésor. Vous disposez de bottes anti-gravité de sorte que les montées ne vous coutent aucun effort. Dans lequel des deux paysages préféreriez-vous intervenir, et pourquoi ?

La réponse paraît évidente. Dans le deuxième paysage, le relief ne vous donne guère d'indice sur la présence éventuelle d'une solution à proximité. Aucune stratégie ne semble pouvoir être plus efficace qu'une exploration au hasard.

Tandis que dans le premier paysage, qui en quelque sorte plus « lisse », en privilégiant les déplacements qui ne nous font pas remonter on peut certes se faire piéger dans un « plateau » constitué de deux extremums locaux, mais si on passe suffisamment près de la solution, on va très vite la trouver.

ATTENTION ! Avec les paysages de recherche des problèmes difficiles, tels que le voyageur de commerce, les choses sont beaucoup plus compliquées. D'une part, un paysage « lisse » est une condition nécessaire mais en aucun cas suffisante à une recherche efficace. Par exemple, il faut que la fonction coût prenne suffisamment de valeurs différentes pour permettre l'existence de sortes d'entonnoirs, appelés **bassins d'attraction**, près des extremums globaux, même si en contrepartie de tels bassins d'attraction peuvent exister pour les extremums locaux.

D'autre part, les paysages usuellement utilisés ne ressemblent en rien aux paysages à deux dimensions que nous avons vu, ni même aux paysages à trois dimensions de nos promenades en montagne. Ce sont des paysages ayant des dizaines, voire des centaines et jusqu'à des millions de dimensions, que le cerveau humain a bien du mal à se représenter. Les illustrations en 2 dimensions sont de pures vues d'artiste uniquement destinées à vous persuader que si le paysage est trop accidenté, aucune stratégie de recherche basée sur le coût ne peut être efficace.

Bonnes pratiques de conception

Il n'existe pas de théorie mathématique permettant de concevoir une application de recherche locale stochastique optimalement efficace pour un problème donné, ni pour prévoir le comportement d'une telle application. Le choix des différents ingrédients, particulièrement l'espace de recherche, le voisinage et le critère de transition, incluant si applicable la fonction coût, ne peut être validé par l'expérimentation. Il y a toutefois quelques **bonnes pratiques** à conseiller. Les respecter ne garantit pas de concevoir une application de recherche locale qui donnera satisfaction, mais ne pas les respecter risque fort de vous conduire à l'échec.

Connexité du graphe de voisinage

Il faut que toute configuration soit connectée à une solution par au moins un chemin dans le graphe de voisinage. Des composantes connexes sans solutions ne sont évidemment pas souhaitables parce que toute recherche commençant ici ne pourrait aboutir.

Densité de solution

Choisir un espace de recherche avec une densité de solutions aussi élevée que possible en respectant les autres bonnes pratiques est souvent un gage d'efficacité, mais on ne peut en faire une règle absolue.

Diamètre du graphe de voisinage

La longueur du plus court chemin entre deux configurations quelconques, et à fortiori entre une configuration quelconque et une solution, doit être très petite au regard de la taille de l'espace de recherche. Autrement dit, il doit exister un chemin permettant d'atteindre rapidement une solution depuis n'importe quelle configuration initiale, même si ce chemin est difficile à trouver. Dans nos expérimentations précédentes sur le voyageur de commerce, pour N villes, l'espace de recherche contient $N!$ configurations (chaque circuit étant représenté par $2N$ configurations différentes) et il est possible d'aller de n'importe laquelle d'entre elles à n'importe quelle autre en au plus N transitions.

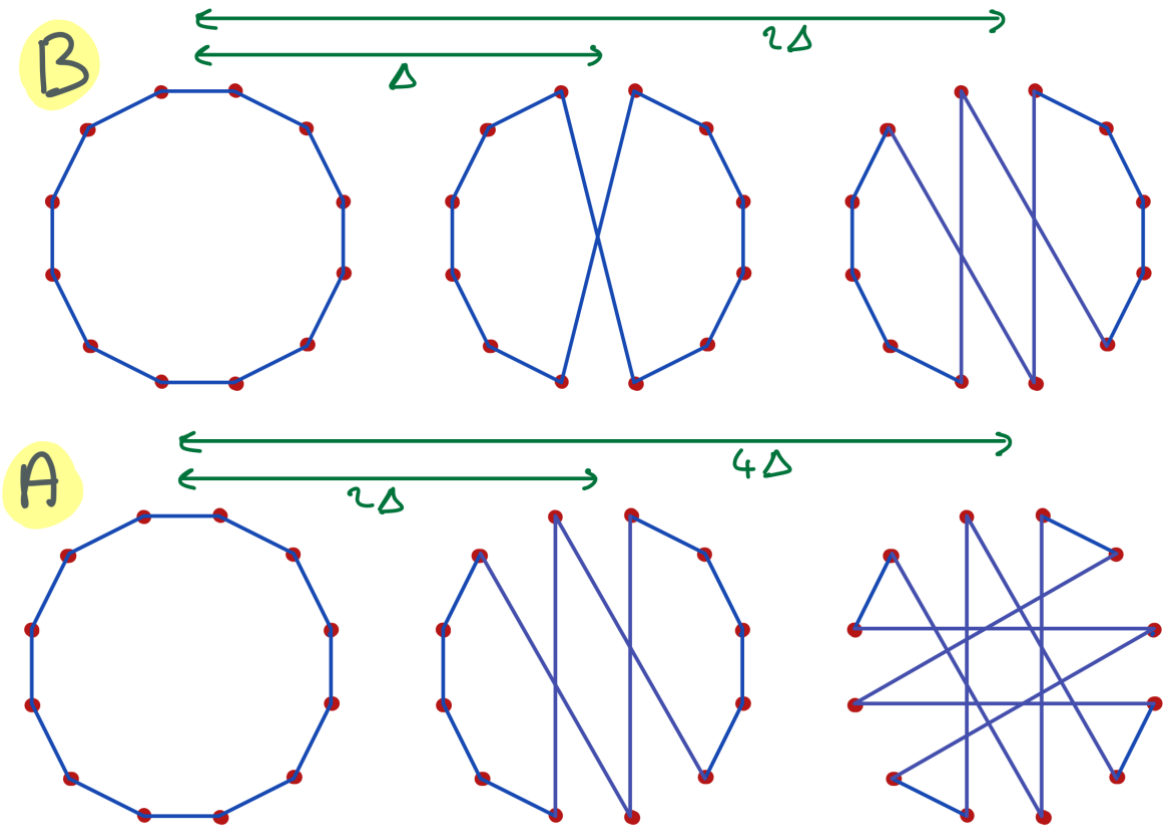
Valeurs de la fonctions coût

La fonction coût doit pouvoir prendre suffisamment de valeurs possibles pour créer un « relief » dans le paysage de recherche. Par exemple une fonction coût qui ne prendrait que deux valeurs : 0 pour les solutions et 1 pour les autres configurations, ne peut pas permettre une recherche plus efficace qu'une marche aléatoire non guidée. Il est souhaitable que la fonction coût puisse prendre un nombre de valeurs différentes au moins du même ordre que le diamètre du graphe.

Un paysage lisse sans être plat

Si de nombreuses configurations interconnectées ont le même coût, le paysage de recherche vont contenir de grands plateaux dans lesquels le coût ne donne aucune information sur la proximité éventuelle d'une solution et encore moins quelle direction prendre pour en trouver une. Mais si les coûts de configurations proches sont aussi peu corrélés que s'ils avaient été déterminés au hasard, ce n'est pas mieux.

Un premier indice simple à vérifier est que la différence de coût entre deux configurations voisines doit être faible au regard de l'amplitude des valeurs de coût, et au regard de l'écart entre le coût des solutions et le coût moyen d'une configuration. On parle parfois de **corrélation voisinage-coût**. Un deuxième indice, parfois formalisé sous le nom de **corrélation distance-coût**, est la tendance à l'augmentation progressive du coût lorsqu'on s'éloigne d'une solution. Pour comparer des voisinages différents avec des configurations et une fonction de coût identiques, comme c'est le cas pour nos voisinages A et B du problème du voyageur de commerce, on peut raisonner en première intention sur une instance très simple à résoudre, comme par exemple des villes disposées sur un cercle.



On voit très bien sur les figures ci-dessus que quand on s'éloigne de la solution, le coût augmente plus vite avec le voisinage A. Or les coûts moyens des circuits aléatoires sont les mêmes pour les deux paysages. Le paysage B est, comparativement, plus lisse, plus « modelé » à proximité des solutions, et d'ailleurs aussi à proximité des extremums locaux, et présente des différences de coûts moins importantes entre configurations voisines, d'une manière plus générale.

Critère de transition

Qu'il soit ou non basé sur une fonction coût, le critère de transition doit permettre **d'intensifier** la recherche dans une partie du paysage où se trouvent les solutions, mais aussi de **diversifier** cette recherche pour éviter de rester bloqué par exemple dans le bassin d'attraction d'un extremum local. Dans le cas de l'utilisation d'une fonction coût, l'intensification se fait en tentant de minimiser le coût. La diversification peut se faire, comme dans le cas du critère de Metropolis, en autorisant occasionnellement des augmentations du coût. Mais il existe d'autres stratégies, comme par exemple la recherche « tabou » dans laquelle on garde suffisamment d'information sur les dernières configurations visitées pour empêcher les transitions qui conduirait à les revisiter. On peut aussi détecter une stagnation, soit du coût, soit de la position de la configuration courante dans l'espace de recherche, et réaliser alors un petit « saut » par plusieurs transitions aléatoires.

Des stratégies beaucoup plus élaborées ont été imaginées. Dans le cas de problèmes où on cherche à satisfaire plusieurs contraintes, on peut par exemple modifier dynamiquement paysage de recherche. A chaque contrainte est assigné un poids, qui peut être par exemple fixé à 1 en début de recherche. Le coût d'une configuration est la somme des poids des contraintes falsifiées. Lorsqu'une contrainte est souvent falsifiée, on augmente son poids, ce qui a pour effet de modifier la forme du paysage de recherche et de favoriser l'exploration de configurations qui satisfont cette contrainte. Les extremums locaux peuvent de déplacer et leurs bassins d'attraction se déformer alors que les solutions, qui ont un coût nul, restent fixes.

Conclusion

La recherche locale stochastique est un moyen simple, facile à implémenter, de résoudre notamment des problèmes de recherche et d'optimisation difficiles à résoudre avec d'autres techniques. De tels problèmes interviennent dans de nombreux domaines couverts par la programmation par contraintes et l'optimisation sous contraintes comme par exemple la logistique, la construction d'emplois du temps, le routage, le câblage de satellite, tomographie discrète etc.

Mais elle a trois gros inconvénients :

1. Il n'y a pas de méthode sûre permettant de choisir les meilleurs ingrédients pour résoudre un problème donné. Il faut soit chercher dans la littérature des travaux ayant déjà été réalisés sur ce problème, soit faire de nombreuses expérimentations pour trouver un espace de recherche, un voisinage et un critère de transition qui donnent satisfaction.

2. Une procédure de recherche locale stochastique ne garantit pas la découverte d'une solution dans un temps compatible avec les besoins des utilisateurs.

3. Si l'instance du problème traité n'a aucune solution, la recherche locale stochastique (dans l'acception du terme utilisée ici) ne permet pas de le prouver. Si la recherche ne se termine pas dans un laps de temps donné, aussi long soit-il, c'est soit parce qu'il n'y a pas de solution, soit parce qu'une solution existe mais n'a pas encore été trouvée.

Il faut toutefois relativiser les deux premiers inconvénients, et même le troisième, en précisant que pour les problèmes visés, les solveurs complets, garantissant de trouver une solution quand il y en a au moins une et capables de déterminer qu'il n'y en a pas dans le cas contraire, sont également difficiles à mettre au point et ont parfois des temps d'exécution tellement prohibitifs qu'en pratique, on est obligé de les arrêter avant de trouver une solution ou d'être sûr qu'il n'en existe aucune.

Exercices d'assimilation

Mode d'emploi

Les exercices proposés peuvent avoir plusieurs solutions. Vous devez d'abord essayer de les faire seul et sans aide.

Si vous êtes bloqué

Vous pouvez d'abord tenter de travailler avec d'autres élèves qui en sont au même stade dans leur progression. Attention, si vous aidez un autre élève et que vous avez de l'avance sur lui, ne lui donnez pas une solution mais des indices tels que ceux proposés dans la section ***Indices*** et des explications. Ensuite, si vous ne pouvez pas travailler en petit groupe ou si le groupe ne parvient pas à proposer une solution, demandez conseil à votre enseignant. Si l'enseignant n'est pas disponible, allez chercher un indice dans la section ***Indices***.

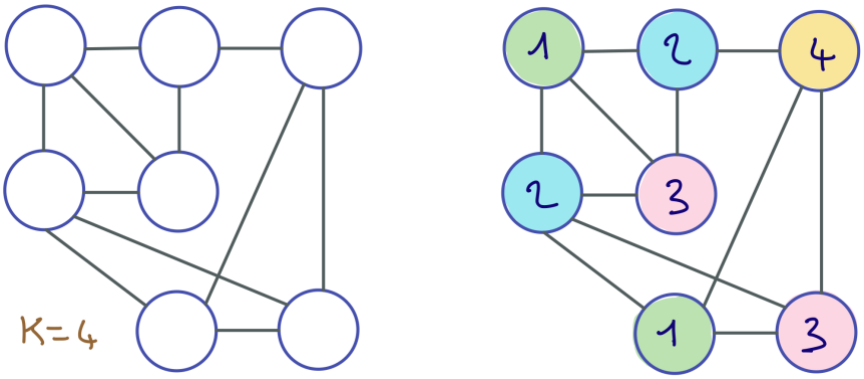
Si vous pensez avoir une solution

Demandez d'abord confirmation à votre enseignant, s'il est disponible, ou consultez les solutions proposées dans la section ***Indices***. En cas de doute, ou si votre solution est différente de celles proposées, demandez confirmation à votre enseignant.

Énoncés

Coloriage de graphe

Une instance de ce problème est un graphe, constitué de sommets reliés par des arêtes, et un entier K. L'objectif est d'attribuer à chaque sommet un entier compris entre 1 et K, appelé *couleur*, de telle sorte que deux sommets reliés par une arête aient toujours des couleurs différentes. Ci-dessous, une instance du problème et une solution de cette instance.

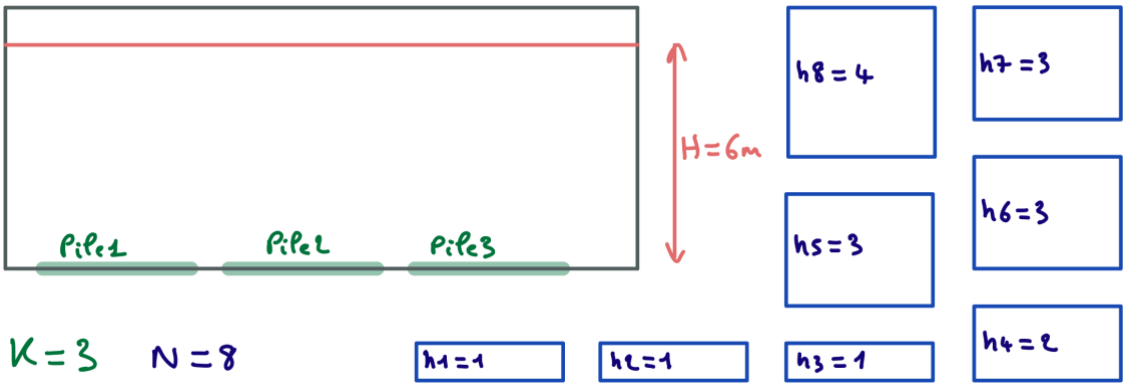


Proposez un **espace de recherche**, un **voisinage** et une **fonction de coût** permettant de traiter ce problème par recherche locale stochastique avec critère de transition de type Metropolis.

Comment modifier l'application de recherche locale stochastique basées sur les critères choisis précédemment pour rechercher un coloriage utilisant un nombre minimal de couleur. On traite donc un problème d'optimisation et non de recherche.

Bin packing 1D

Dans un entrepôt, on peut empiler des caisses jusqu'à une hauteur H. Il y a N caisses à ranger en K piles. Elles ont chacune une hauteur spécifique $(h_1, \dots, h_N)$ mais leurs autres dimensions sont identiques. Voici un *exemple*.



On veut trouver un moyen de ranger toutes les caisses en utilisant un algorithme de recherche locale stochastique avec critère de transition de type Metropolis. Proposez un **espace de recherche**, un **voisinage** et une **fonction de coût**.

Problème du sac à dos

On se trouve dans une pièce où il y a N objets de poids différents : $p_1, \dots, p_N$. Ces objets ont des valeurs différentes $v_1, \dots, v_N$. Les unités n'ont pas d'importance. Les poids pourraient par exemple être exprimés en kilogrammes et les valeurs en Euro. On cherche à sélectionner certains de ces objets pour les mettre dans un sac à dos. Le poids total des objets choisis ne doit pas excéder une valeur P donnée et leur valeur totale doit être au moins égale à une valeur V donnée. Il n'y a pas de contrainte sur le nombre d'objets à rassembler.

Proposez un **espace de recherche**, un **voisinage** et une **fonction de coût** permettant de traiter ce problème par recherche locale stochastique avec critère de transition de type Metropolis.

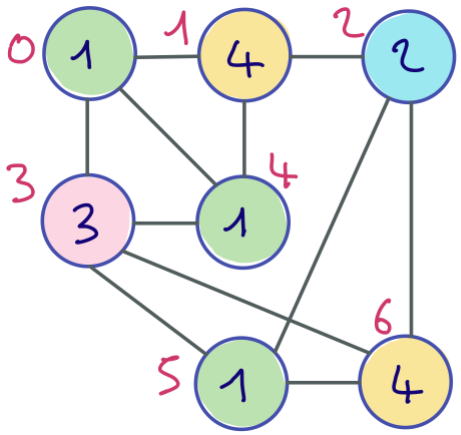
Considérez ensuite le problème d'optimisation dans lequel la contrainte de poids est identique, mais on cherche à maximiser la valeur totale des objets à rassembler. Quelles modifications proposez-vous de faire au modèle précédent pour traiter cette variante du problème ?

Indices

Coloriage de graphe

Indice 1

Une configuration de **l'espace de recherche** représente un coloriage, c'est-à-dire l'assignation d'une couleur à chaque sommet du graphe. Cela peut se présenter comme un simple tableau. On numérote les sommets du graphe à partir de 0 et, dans le tableau, la cellule d'indice i contient un entier qui représente la couleur du sommet i. L'espace de recherche contient tous les coloriages possibles.



Indice 2

Le **coût** d'une configuration est le nombre d'arrêtes reliant deux sommets de même couleur. Le coloriage est correct si et seulement si cette valeur est 0. Dans l'exemple ci-dessus, le coût vaut 1.

Indice 3

Voisinage simple, à essayer en première intention : on passe d'une configuration à une de ses voisines en changeant la couleur d'un sommet. Pour graphe de N sommets à colorier avec K couleurs, chaque configuration a donc $N(K-1)$ voisines.

Indice 4

Le nombre de couleurs n'est pas imposé mais on cherche à le minimiser. On sait qu'un coloriage en N couleurs est toujours possible. On peut donc décider que tout coloriage avec un nombre de couleurs entre 2 et N est une configuration. Pour réduire les possibilités, on peut autoriser uniquement les coloriages avec les couleurs $\{1,2\}$, $\{1,2,3\}$, $\{1,\dots,N\}$. Une configuration utilisant les couleurs $\{1,\dots,k\}$ a pour voisine toute configuration dans laquelle la couleur d'un sommet a été remplacée par une autre dans l'intervalle $1..\min\{N,k+1\}$.

Il faut bien sûr que le nombre total de couleurs utilisées intervienne dans le coût. On peut par exemple définir le coût d'une configuration C comme la somme de deux termes $E(C)+\alpha\,K(C)$, où $E(C)$ est le nombre d'arrêtes reliant des sommets de même couleur et $K(C)$ est le nombre de couleurs utilisées.

Une autre approche consiste à utiliser la recherche locale stochastique telle que décrite par les indices 1 à 3, successivement avec $N, N-1, N-2, \dots$ couleurs jusqu'à ce qu'on ne parvienne plus à trouver de solution avec le quota de temps de calcul dont on dispose.

Bin packing 1D

Indice 1

Les caisses sont identifiées par les entiers 1, …, N. Une **configuration** $\langle P_1, \dots P_K\rangle$ est une K-partition de l'ensemble {1, 2, …, N}, au sens où $P_1, \dots P_K$ sont des ensembles disjoints dont l'union est {1, 2, …, N}. Peu importe le détail de la représentation en mémoire d'une telle configuration. Cela peut être par exemple K tableaux ou listes. Chacun des K ensembles représente une pile de caisses (l'ordre des caisses dans la pile n'a pas d'importance).

Indice 2

Les **voisines** d'une configuration $\langle P_1, \dots P_K\rangle$ sont obtenues en retirant un élément (si applicable) de $P_i, i \in 1..K$ et en l'ajoutant à un autre ensemble $P_j, j \in 1..K, i \neq j$.

Indice 3

Le **coût** d'une configuration $\langle P_1, \dots P_K\rangle$ peut être définit comme la somme des débordements $D(P_1)+\dots+D(P_K)$, où $D(P)$ est calculé de la manière suivante : soit $S(P)=\sum_{e\in P} h_e$ la somme des hauteurs des caisses de P.

- si $S(P) \leq H$ alors $D(P)=0$,
- sinon $D(P)=S(P)-H$.

Sac à dos

Indice 1

Une configuration est un ensemble $C \subseteq 1..N$ qui contient les identifiants des objets sélectionnés. On peut le représenter par une liste où un tableau de bits, par exemple.

Indice 2

Si pour toute configuration C on note, $p(C) = \sum_{i \in C} p_i$ et $v(C) = \sum_{i \in C} v_i$, les deux critères à respecter pour que C soit une solution sont $p(C) \leq P$ et $v(C) \geq V$. On peut utiliser comme coût de C quelque chose comme…

Indice 3

Proposition de coût d'une configuration C :

$$\alpha \max\{V - v(C), 0\} + \beta \max\{p(C) - P, 0\}$$

où α et β sont des coefficients à déterminer expérimentalement.

Indice 4

On veut maximiser la valeur en respectant la limite de poids. L'approche qui me parait la plus simple consiste à restreindre l'espace de recherche aux seules configurations qui respectent la contrainte de poids, que nous appellerons configurations valides, et de baser le coût uniquement sur la valeur des objets sélectionnés. Par exemple, on pourrait définir le coût d'une configuration C par $1/v(C)$ ou encore $-v(C)$.

Mais il est possible que ce choix complique l'accès à une solution optimale depuis certaines configurations. Au lieu d'interdire les configurations non valides en les retirant de l'espace de recherche, on pourrait par exemple agir sur leur coût en lui ajoutant une constante suffisamment grande pour qu'une configuration de coût minimum soit nécessairement valide. Mais ceci nous amène au-delà de la simple initiation à la recherche locale stochastique…