Heuristiques

Nicolas Bousquet

January 22, 2024

Jusqu'à présent on a vu des techniques algorithmiques qui permettait de résoudre optimalement des problèmes. Autrement dit, on garantissait qu'on trouvait la meilleure solution possible. Malheureusement pour certains problèmes, de tels algorithmes nécessite un temps d'execution exponentiel. On peut alors se demander si on peut trouver une solution raisonnablement bonne avec un temps d'éxecution décent. Algorithmes d'approximation et heuristiques ont ce but. Mais l'approche est différente:

- Un algorithme d'approximation est un algorithme à *garantie de performance*. Autrement dit, on peut garantir que la solution n'est pas (trop) éloignée d'une solution optimale.
- Une heuristique est un algorithme sans garantie de performance qui fonctionne bien "en pratique". Il existe des heuristiques pour lesquelles on ne sait pas evaluer leur efficacité théorique. Pour certaines, on peut montrer qu'il existe des instances où elles se comportent arbitrairement mal (mais ces instances n'arrivent pas "en pratique"). Le choix de l'heuristique dépend donc assez souvent des instances, certaines se comportant mieux que d'autres selon la structure des instances.

1 Heuristiques

Dans cette partie notre problème de référence sera le problème de l'indépendant maximum, c'est à dire trouver un ensemble de sommets deux à deux extrémités disjointes qui est de taille maximum.

1.1 Recherche locale et ses variantes

Graphes de configurations. Il existe un algorithme extrèmement simple pour trouver un indépendant maximum dans un graphe. On va considérer pour cela le graphe des configurations pour cela.

Le graphe des configurations de G est un (meta)graphe dont l'ensemble des sommets est l'ensemble des ensembles indépendants de G. Et on va rajouter une arête entre un ensemble I et un ensemble J si et seulement si I et J ne diffèrent que d'un seul sommet (autrement dit I peut être obtenu depuis J par un ajout de sommet ou le contraire).

Comme on peut atteindre tous les sommets du graphe de configurations depuis le sommet $\emptyset$ avec un chemin de longueur au plus n, le graphe de configurations est connexe. Donc pour trouver un ensemble indépendant maximum il suffit de calculer un arbre couvrant du graphe de configurations et renvoyer le plus petit ensemble indépendant qui aura été trouvé.

Evidement, cet algorithme ne fonctionne pas du tout: comme le nombre de solutions est exponentielle, il faudra un temps exponentiel pour parcourir le graphe de configurations (et en plus, il faudra un espace exponentiel !). Donc cette approche en toute généralité est vouée à l'échec.

Algorithmes gloutons et graphes de configurations. Cependant, on peut se dire qu'au lieu de faire un parcours complet de ce graphe, on pourrait seulement essayer de parcourir les voisins dans le graphe de configurations qui nous paraissent intéressants. Il existe deux approches naturelles: une d'elle consiste à faire une marche aléatoire sur le graphe de configurations. Malheureusement, cette approche, bien qu'elle finisse presque sûrement par trouver la solution optimale converge extrêmement lentement.

Une seconde approche que l'on a déjà vu est celle des algorithmes gloutons. Pour un algorithme glouton, on part de la solution $\emptyset$ et on rajoute un sommet à la solution tant que possible. Autrement

dit, on part du sommet $\emptyset$ et on se déplace dans le graphe de configurations vers un "sommet" qui correspond à un ensemble de taille 1, puis à un ensemble de taille 2...etc... Et on augmente la taille de notre ensemble tant que possible.

Le problème de cette approche est qu'on est rapidement bloquée dans un optimum local qu'on ne peut pas améliorer (un ensemble maximal par inclusion). Mais cet optimum local est peut être arbitrairement loin d'un optimal global. Par exemple dans le cas d'une étoile, on peut être bloqué dans une solution de taille 1 alors que la solution peut être arbitrairement grande !

Rechercle locale Les algorithmes de recherche locale consiste a essayer de réparer les algorithmes gloutons en essayant de se dire qu'on a peut être fait une (ou plusieurs) petites erreurs localement. Et le but sera de les repérer et des les corriger. On va donc "augmenter" un peu nos opérations localement

On peut donc sortir de la solution optimale locale en remplaçant un sommet par un meilleur ensemble. Par exemple dans le cas de l'étoile, cet algorithme nous permettra de sortir de la solution "idiote" en renvoyant une solution optimale. Cependant on a un problème: comment trouver la meilleure completion possible de $I \setminus x$? En effet ce problème est potentiellement aussi difficile que le problème initial. Cepdant, il suffit de trouver une meilleure solution pour être déjà satisfait dans notre cas. Autrement dit, si, on peut compléter l'ensemble indépendant avec deux sommets supplémentaires alors notre algorithme de recherche locale améliore la solution et cela nous rend heureux.

Cependant, dans le cas d'un biparti complet avec deux sommets d'un coté et n de l'autre, cet algorithme ne nous permettra pas de trouver la solution optimale. On peut décider de supprimer plusieurs sommets au lieu d'un seul et tenter d'améliorer la solution. Cependant cela a un coût: c'est de plus en plus difficile de tester s'il existe une solution meilleure une fois qu'on a enlever ces sommets !

Si on se place dans le graphe de configuration, cela signifie que l'on modifie les arêtes du graphe. En effet, maintenant, on aura un graphe orienté où on peut remplacer I par J tel que $|J| > |I|$ et J contient $I \setminus x$ pour un certain $x \in I$.

Un algorithme de recherche locale est donc un *algorithme de descente* et on peut l'exprimer ainsi:

- choisir une solution $s \in S$;
- Déterminer une solution s_0 qui minimise f dans $N(s)$ dans le graphe de configurations;
- si $f(s_0) < f(s)$ alors poser $s \leftarrow s_0$ et retourner au point précédent, sinon retourner s.

Vu d'un très haut niveau, on peut donc toujours considérer qu'un algorithme de recherche locale est un algorithme glouton sauf que les opérations de modifications qu'on s'autorise à faire sont plus générale que les opérations triviales (rendant l'algorithme plus complexe, plus long et l'analyse plus compliquée !).

Recuit simulé Une des variantes les plus classiques de la recherche locale est le recuit simulé. Cette métrode a été introduite par Kirkpatrick, Gelatt, Vecchi en 1983. Dans le cas du recuit simulé, on va s'autoriser à dégrader partiellement la solution courante pour essayer de trouver une meilleure branche. L'idée est la suivante: peut être que l'optimum local est "assez profond" et par conséquent on a envie de pouvoir sortir d'un optimum local s'il n'est pas bon. On va donc, à chaque étape, s'autoriser à sélectionner une solution "moins bonne" que celle actuelle, mais on a envie de le faire avec un probabilité qui diminue:

- En fonction de la degradation de la solution (si on dégrade peu la solution, c'est mieux !)
- En fonction du nombre d'étapes depuis lequel on a commencé (on s'autorise plus facilement des degradations en début d'algorithme).

Plus formellement, l'algorithme de recuit simulé s'exprime en général ainsi:

- Choisir une solution $s \in S$ ainsi qu'une température initiale T;
- Tant que aucun critère d'arrêt n'est satisfait faire
- Choisir aléatoirement une solution $s_0 \in N(s)$;

- Générer un nombre réel aléatoire $r \in [0,1]$; Si $r < p(T,s,s0)$ alors poser $s \leftarrow s_0$; Mettre à jour T; Fin du tant que

En général on pose $p(T,s,s_0) = e^{-\frac{f(s)-f(s_0)}{T}}$. Cela permet:

- D'accepter avec probabilité 1 si $f(s_0) < f(s)$, c'est à dire si on améliore la solution.
- D'accepter avec un probabilité qui diminue rapidement si la qualité de la solution devient très mauvaise par rapport à la solution actuelle.

La mise à jour de la température se fait typiquement par un facteur multiplicatif à chaque étape. On pose par exemple $T \leftarrow 0.95T$.

Alors qu'il n'existe aucune garantie globale de l'algorithme de descente, on peut montrer que le recuit simulé converge avec grande probabilité vers une solution optimale sous certaines conditions. Aarts, Korst et Laarhoven ont démontré en 1997 que sous certaines conditions, l'algorithme du Recuit Simulé converge en probabilité vers un optimum global lorsque le nombre d'itérations tend vers l'infini. Plus précisément, soit $P(k)$ la probabilité que l'optimum global soit trouvé après k itérations, et soit T_k la température à la k e itération. Les auteurs susmentionnés ont démontré qu'il existe $L \in R$ tel que

$$\lim_{k \to +\infty} P(k) = 1 \Leftrightarrow \sum_{k=1}^{\infty} e^{L/T_k} = +\infty$$

En particulier si on prend une température qui décroit lentement ($\approx 1/\log(k)$ par exemple), cette propriété est vérifiée. Malheureusement, le temps de convergence rend ce résultat théorique innaplicable en pratique, d'où une descente de température plus rapide.

Il existe cependant de nombreuses stratégies de modifications de température en pratique. Par exemple

- Modification par paliers (comme on le verra en TP) - on ne modifie la température qu'après un certain nombre de tours.
- Modification non monotone. (On peut "reaugmenter" la température si on est bloqué depuis longtemps dans un optimum local).

Recherche tabou La recherche tabou est une autre variante de la recherche locale avec deux petites modifications:

- On peut ne pas améliorer la qualité de la solution mais seulement l'égaler (on peut se déplacer sur un palier)
- On retient une liste de configurations qu'on a visité récement et qu'on ne veut pas revisiter.

Typiquement, on fixe un entier p et on s'interdit de revenir dans les p dernières positions. Cela nous permet d'interdire de revisiter encore et encore les mêmes solutions d'un plateau et favorise donc son exploration totale. Cette méthode a été introduite par Glover en 1986.

Illustrons la recherche tabou sur un exemple extreme. Imaginons par exemple que le graphe de configuration soit un cycle de longueur n. Toutes les solutions ont valeur 1 sauf la solution $n/2$ qui a valeur 0. Si on applique une recherche locale simple (en ayant le droit de se déplacer sur le plateau), on va faire une marche aléatoire sur le cycle. Le temps d'atteindre le sommet $n/2$ est alors assez long. Au contraire, dans le cas de la recherche tabou, on va choisir une direction sans faire marche arrière et atteindre le point $n/2$ en $n/2$ étapes depuis le point 0.

Encore une fois, de nombreuses variantes existent: liste dont la taille varie avec le temps ou adaptative en longueur selon le temps depuis lequel la solution n'a pas été améliorée.

Autres méthodes Il existe pleins d'autres variantes de recherche locale: GRASP (on construit petit à petit des solutions en partant de sous ensembles), recheche à voisinages variables...etc...

1.2 Algorithmes evolutifs

Description générale L'idée est d'appliquer la théorie du darwinisme à l'algorithmique. Le problème des méthodes précédentes est qu'on ne modifie qu'une seule solution. On n'a donc aucune "mixité génétique". Cet individu évolue donc vers un optimum local mais, s'il n'est pas global, il n'a pas la diversité génétique nécessaire pour s'adapter et s'améliorer pour trouver une meilleure solution.

L'idée des algorithmes évolutifs est de palier à ce problème en faisant évoluer une population d'individus selon des règles bien précises. Ces méthodes alternent entre des périodes d'adaptation individuelle et des périodes de coopération durant lesquelles les individus échangent de l'information. Dans les grandes lignes, un algorithme évolutif s'exprime ainsi:

- Débuter avec une population initiale d'individus ;
- Tant que aucun critère d'arrêt n'est satisfait faire
- Exécuter une procédure de coopération (aussi appelé reproduction);
- Exécuter une procédure d'adaptation individuelle (aussi appelé mutation);
- Fin du tant que

La phase de reproduction consiste a essayer de recombiner les solutions entre elles pour essayer de concevoir une meilleure solution. La phase de mutation elle va tenter d'améliorer localement la solution obtenue (voir la réparer si ce n'est pas tout à fait une solution). Il peut arriver que les algorithmes évolutifs ne propose pas de mutation. Autrement dit on va simplement Il y a, ensuite, pleins de possiblités pour concevoir un tel algorithme:

- Types d'individus. On ne manipule pas forcement des solutions dans les algorithmes évolutifs. Par exemple on peut imaginer des solutions partielle à un problème. Par exemple, pour des tournées de véhicules, un individu peut être la tournée d'un véhicule qui visite un sous-ensemble de clients alors qu'une solution est un ensemble de tournées (et donc d'individus) qui visitent tous les clients.
- Evolution de la population. A chaque itération la population évolue. Il est naturel de se poser la question de quels sera la population a l'étape suivante. Comme la population ne peut pas augmenter indéfiniment, il faut choisir quels individus survivront aux étapes suivantes (en général la solution reste de taille fixe):
  Seulement les enfants (nouvelles solutions)? On parle alors de remplacement générationnel.
  A la fois des enfants et des parents? On parle alors de remplacement stationnaire.
  En général la taille de la population est fixe et on garde, selon le choix qu'on fait, les p meilleures solutions (même si on peut en générer bien plus que p).
- Sources de la reproduction. Comment d'invidus sont nécessaires pour générer une nouvelle solution? En général, on a besoin de deux sources, que l'on appelle parents et les solutions générées sont appelées enfants. (Cependant dans certains cas, les parents ne suffisent pas, on rediscutera un peu plus tard).
- Reproduction. On a dit qu'on devait combiner les solutions. Une question naturelle qui se pose est donc de savoir avec qui les individus de notre population ont le droit de se reproduire. Soit on impose aucune contrainte et on a donc un reproduction non structurée. Dans le cas contraire, on a une reproduction structurée et on represente les reproductions possibles avec un graphe qui est la plupart du temps soit un cycle (un individu porura se reproduire avec deux individus) soit une grille (un individu peut se reproduire avec 4 autres individus).
- Irréalisabilité. Que doit-on faire des solutions non réalisables générées? Les éliminer? Les garder avec une très faible probabilité?
- Intensification. Lorsqu'on peut localiser l'information pertinente qui rend un individu meilleur qu'un autre, il faut développer des procédures de coopération qui créent des enfants en combinant adéquatement les informations pertinentes de chacun des parents. (Un peu comme on le

ferait en sélection génétique en laboratoire). Dans ce cas la phase d'adaptation individuelle n'est alors pas indispensable. Mais dans de nombreux cas, il est difficile de reperer une telle information et la solution générée peut à la fois avoir hérité de propriétés intéressantes mais également de propriétés complètement inutiles. On doit alors appliquer un algorithme d'adaptation indivuduelle pour effacer les propriétés non-pertinentes tout en gardant les caractéristiques intéressantes. Typiquement, on fait cela avec un algorithme de recherche locale.

- Diversification. Parfois les algorithmes de recherche locale on tendance à converger rapidement vers un ensemble d'indivudus très semblables. Il existe des solutions pour éviter de tomber dans ce genre d'ecueils (ajout de bruits dans les solutions...etc...).

Algorithmes génétiques. L'algorithme génétique introduit par Holland en 1962, Rechenberg (1965) et Fogel et al, 1966 est la version la plus simple et naturelle des algorithmes évolutifs.

- Générer une population initiale P_0 de $p \geq 2$ individus ;
- Pose $i = 0$;
- Tant que aucun critère d'arrêt n'est satisfait faire
- $i \leftarrow i + 1$;
- Répéter p fois les 2 lignes suivantes
- Créer une enfant E en croisant 2 individus I_1 et I_2 de P_{i-1} ;
- Appliquer une opérateur de mutation à E et rajouter l'enfant ainsi modifié à P_i ;
- Fin du tant que

On peut aussi décrire la méthode avec les caractéristiques vues précédement:

- Types d'individus. Solutions
- Evolution de la population. Remplacement générationnel
- Sources de la reproduction. 2 parents
- Reproduction. Population structurée ou non structurée.
- Irréalisabilité. On ne génère que des solutions
- Intensification. Aucune
- Diversification. Aucune

Variantes Ce type de méthodes fonctionnant assez bien dans de nombreux cas, de nombreuses variantes ont vu le jour.

Sources. Notes de cours sont très largement inspirées par celle d'Alain Hertz (GERAD, Université de Montréal) pour la partie méta-heuristiques.