Chapitre 6
Les stratégies de recherche

Claude-Guy Quimper

Introduction

- Nous présentons dans ce chapitre des heuristiques de recherche qui permettent d'accélérer la résolution de problèmes.
- Nous présentons ensuite des techniques qui permettent de diversifier la recherche et d'explorer plusieurs sections de l'arbre de recherche.

La robustesse d'un solveur

- La **robustesse** d'un solveur est sa capacité à résoudre une large variété de problèmes.
- Un solveur peut être configuré pour être très efficace à résoudre un problème en particulier. Il peut également être configuré pour être le plus robuste possible.
- Nous présentons dans ce chapitre des techniques rendant un solveur plus robuste. Ce sont aussi des techniques qui peuvent être appliquées sur des problèmes spécifiques.

Heuristiques de choix de variable

- Il est préférable d'instancier en premier les variables qui ont le plus d'impact sur une solution.
- Dans un problème de placement d'objets, on place les plus gros objets dans la boîte avant de placer les plus petits.
- Dans un problème d'optimisation, on choisit la variable qui a un plus grand impact sur la valeur de la fonction objectif.

Ordre statique

- Lorsque l'on sait quelles variables sont les plus importantes dans le problème, on peut fixer, avant même le début de la recherche, l'ordre dans lequel les variables seront instanciées.
- On parle alors d'un ordre d'instanciation des variables *statique*.

Le plus petit domaine

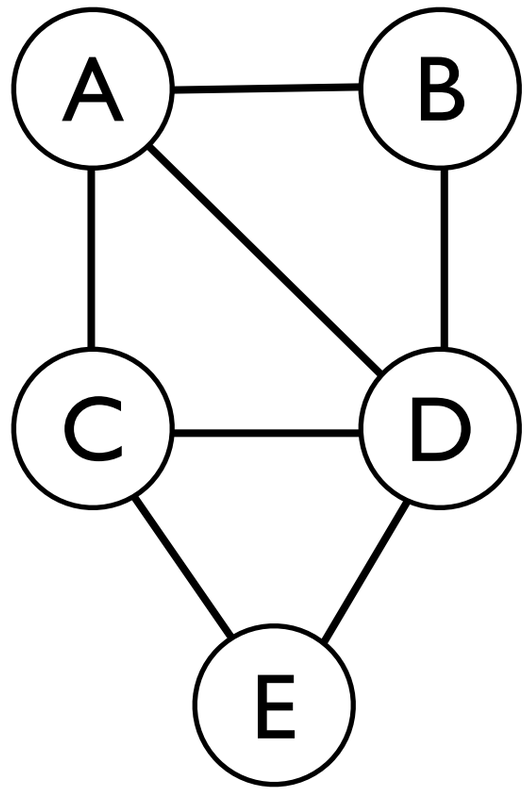
- L'ordre d'instanciation des variables peut être décidé à la volée durant la recherche.
- C'est le cas de l'heuristique qui choisit d'instancier la variable avec le plus petit domaine.
- Ce choix s'appuie sur deux principes:
  - Une variable avec un petit domaine est plus difficile à instancier, car il y a moins de choix pour cette variable.
  - La probabilité de tomber sur le bon choix est plus grande lorsqu'on a moins de choix. De plus, lors d'un retour arrière, on élimine un plus grand pourcentage de l'espace de recherche.
- Cette heuristique est efficace pour un très grand nombre de problèmes et est souvent choisie comme heuristique par défaut dans les solveurs.

La variable la plus contrainte

- Le degré d'une variable dans un problème de satisfaction de contraintes est le nombre de contraintes qui possèdent cette variable dans leur portée.
- On peut supposer qu'une variable avec un grand degré est une variable qui a beaucoup d'impact sur la solution.
- En instanciant ces variables en premier, on s'assure que les algorithmes de filtrage réduiront au maximum l'espace de recherche dès le début de la recherche.

Les choix structurels

- Voici le graphe de contraintes d'un problème.
- **Question**: Qu'arrive-t-il si on instancie D?
- **Réponse**: La variable D devient une constante. Le problème résultant a une structure d'arbre et peut être résolu en temps polynomial.



Les choix structurels

- Certaines études démontrent qu'il est préférable de faire des choix non pas en fonction de la probabilité d'obtenir une solution, mais plutôt en fonction de la probabilité d'obtenir un problème plus simple à résoudre[1].
- Les choix structurels essaient de réduire la largeur d'arbre d'un graphe de contraintes jusqu'à ce qu'il soit soluble en temps polynomial.
- Généralement, les variables à instancier en premier sont identifiées manuellement puisqu'il serait trop lent d'avoir un algorithme identifiant ces variables à la volée.

[1] Certaines études prouvent exactement le contraire. Débrouillez-vous!

Combinaisons d'heuristiques

- On utilise généralement des combinaisons d'heuristique de choix de variables.
- Par exemple, on peut brancher sur les variables X, Y, Z en premier pour des raisons structurelles et ensuite brancher sur les variables ayant les plus petits domaines. Si plus d'une variable a le plus petit domaine, alors on brise l'égalité en branchant sur la variable ayant le plus grand degré.

Les heuristiques stochastiques

- Une heuristique peut faire appel à un générateur de nombres pseudo-aléatoires afin de prendre une décision.
- C'est souvent de cette façon qu'on brise l'égalité lorsque plusieurs choix sont équivalents par rapport à un critère d'évaluation.
- Par exemple, si deux variables ont le plus petit domaine dans le problème, ces deux variables se qualifient pour être choisies par l'heuristique du plus petit domaine. On sélectionne donc au hasard parmi ces deux variables.
- L'ajout d'heuristiques stochastiques diminue considérablement le temps en pire cas pour résoudre un problème.

Apprentissage d'une heuristique

- Il existe des heuristiques qui s'adaptent au problème.
- Ces heuristiques apprennent quelles sont les variables du problème qui ont le plus de difficulté à satisfaire les contraintes.
- Une de ces heuristiques est la suivante.
- Chaque variable est initialisée avec un poids de 0.
- Chaque fois qu'un algorithme de filtrage provoque un retour arrière, les poids des variables dans la portée de la contrainte sont augmentés de 1.
- L'heuristique branche sur la variable ayant le plus grand poids.

Apprentissage d'une heuristique

- Certains solveurs ont des heuristiques adaptatives très complexes. Souvent, les algorithmes « plongent » dans l'arbre de recherche dans le seul but de découvrir de l'information sur les contraintes et les variables et de calibrer les poids sur les variables.

- Les solveurs SAT utilisent presque tous des heuristiques qui apprennent la structure du problème.

Le choix d'une valeur

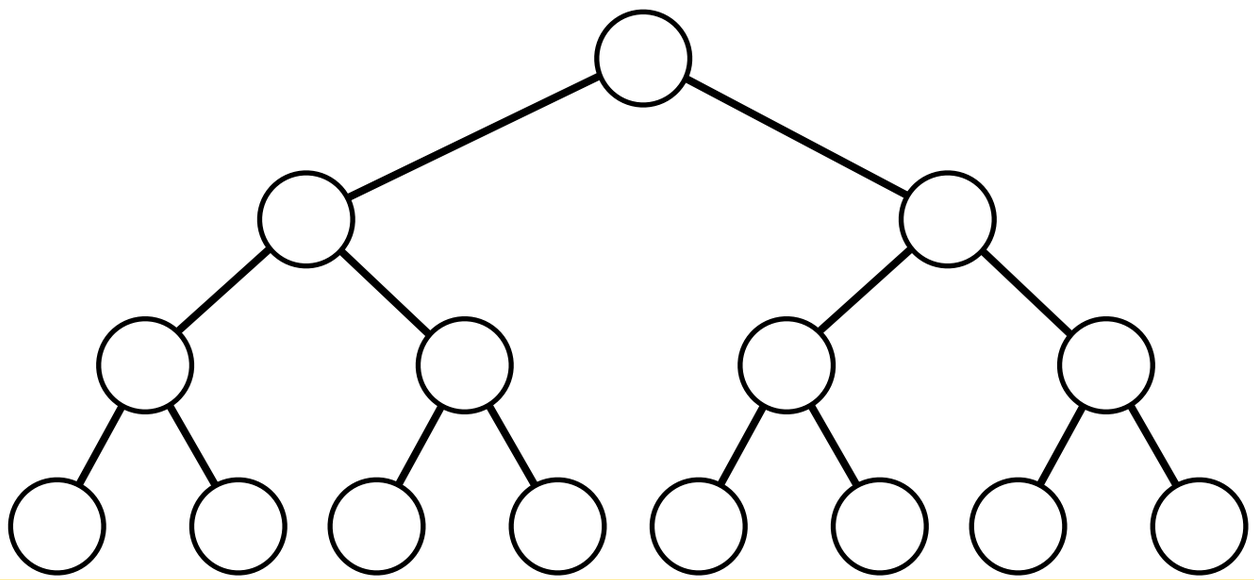
- Une fois que la variable est choisie, il faut décider quelle valeur lui sera affectée.
- On choisit généralement la valeur qui a de meilleures chances de mener à une solution ou la valeur qui optimise la fonction objectif.
- Par exemple, si la fonction objectif est de minimiser 8X - 3Y, on choisit les plus petites valeurs possible pour X et les plus grandes possible pour Y.
- Le choix d'une valeur dépend souvent d'une connaissance du problème.
- Les valeurs sont souvent triées du premier choix au dernier choix.

Les concordances et déviations

- Lorsque l'algorithme de recherche branche sur une variable pour la première fois, l'heuristique de choix de valeurs lui assigne son premier choix. Ce choix est donc en **concordance** avec l'heuristique.
- Lorsque l'algorithme de recherche doit revisiter le choix de valeurs fait pour une variable, la valeur affectée à la variable est donc un deuxième, troisième, ou n ième choix, mais ce n'est pas un premier choix. On dit que ce choix est **dévié** par rapport à l'heuristique.

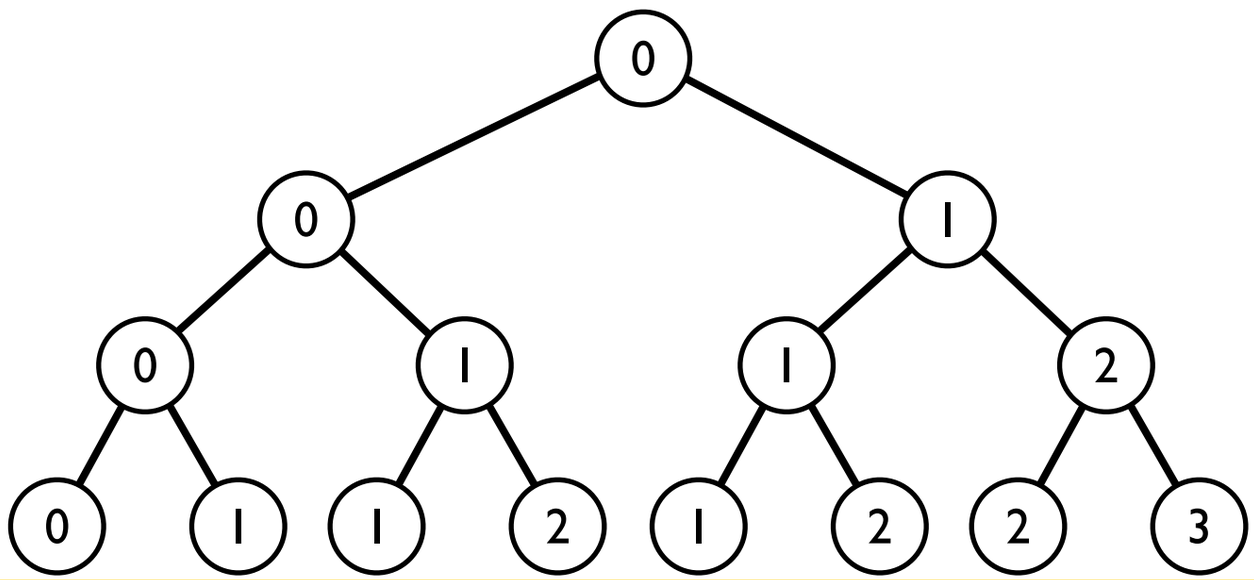
Les déviations dans l'arbre de recherche

- Voici un arbre de recherche. Une fouille en profondeur visite les noeuds de gauche en premier.
- Un branchement vers la gauche est donc concordant et un branchement vers la droite est dévié.



Les déviations dans l'arbre de recherche

- Le nombre de déviations depuis la racine est indiqué dans chaque noeud.

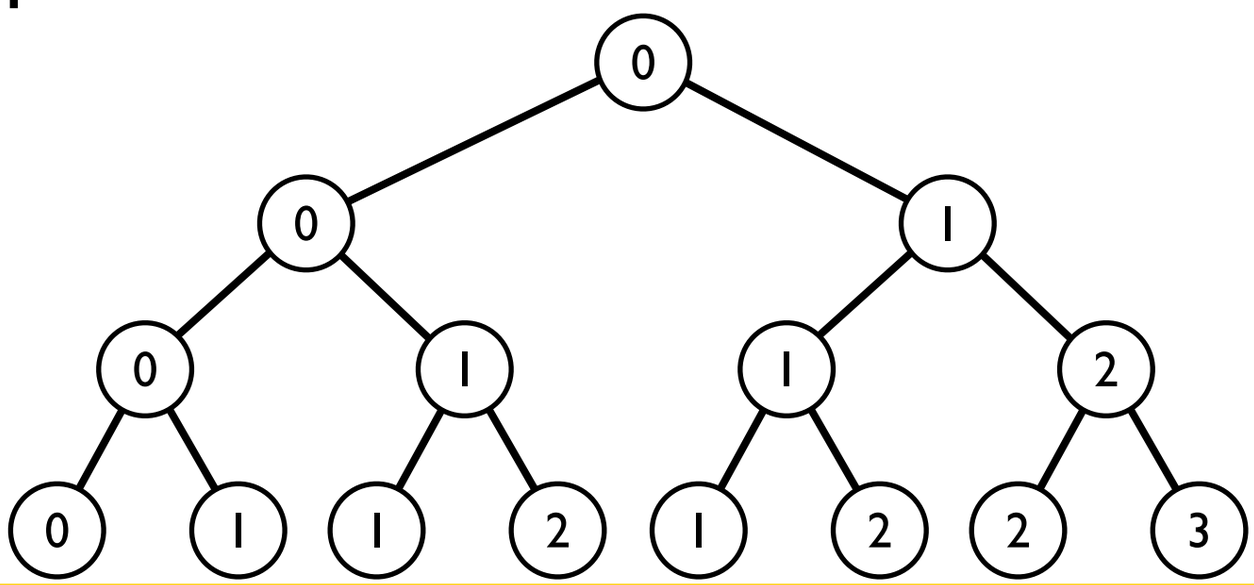


Les déviations dans l'arbre de recherche

- L'heuristique de recherche nous indique que plus le nombre de déviations est petit, plus la probabilité est grande qu'une feuille soit la solution recherchée.

- La fouille en profondeur visite les feuilles de gauche à droite.

- Il serait pourtant plus efficace de visiter en premier les noeuds ayant le plus petit nombre de déviations.



La fouille LDS

(Limited discrepancy search)

- Cette fouille visite les noeuds avec le plus petit nombre de déviations en premier.
- Nous présentons le pseudo-code lorsque les domaines ont une cardinalité de 2.
- L'algorithme peut être généralisé pour des domaines plus grands.
- Lorsque les heuristiques de choix de variables et choix de valeurs sont bonnes, LDS peut radicalement accélérer les temps de calcul.

Algorithme 1 : Fouille-LDS(dom(X_1), ..., dom(X_n))

pour $k = 0..n$ **faire**
  $Solution \leftarrow$ LDS(dom(X_1), ..., dom(X_n), k)
  si $Solution \neq \emptyset$ **alors**
    retourner $Solution$
retourner $\emptyset$

Algorithme 2 : LDS(dom(X_1), ..., dom(X_n), nbDéviations)

$Candidats \leftarrow \{X_i \mid |\text{dom}(X_i)| > 1\}$
si $Candidats = \emptyset$ **alors**
  // Tous les domaines ne contiennent qu'une seule valeur.
  si *dom*(X_1), ..., *dom*(X_n) *satisfait toutes les contraintes* **alors**
    retourner dom(X_1), ..., dom(X_n)
  sinon
    retourner $\emptyset$
Choisir une variable $X_i \in Candidats$
Il y a deux valeurs dans dom(X_i). Soit v_1 et v_2 le premier et le deuxième choix d'après l'heuristique.
si *nbDéviations* $< |Candidats|$ **alors**
  $Solution \leftarrow$ LDS(dom(X_1), ..., dom(X_{i-1}), $\{v_1\}$, dom(X_{i+1}), ..., dom(X_n), nbDéviations)
  si $Solution \neq \emptyset$ **alors**
    retourner $Solution$
si *nbDéviations* > 0 **alors**
  $Solution \leftarrow$ LDS(dom(X_1), ..., dom(X_{i-1}), $\{v_2\}$, dom(X_{i+1}), ..., dom(X_n), nbDéviations $- 1$)
  si $Solution \neq \emptyset$ **alors**
    retourner $Solution$
retourner $\emptyset$

Les effets de LDS

- LDS a la propriété de visiter plusieurs sections de l'arbre de recherche.
- Ainsi, les solutions partielles explorées peuvent être très différentes les unes des autres.
- Dans une fouille en profondeur, il faut visiter la moitié de l'arbre de recherche avant de reconsidérer la valeur affectée à la première variable.

Les redémarrages

- Il existe d'autres techniques pour diversifier la nature des solutions explorées.
- Les redémarrages consistent à utiliser des heuristiques stochastiques. Après « un certain temps » à chercher une solution, on redémarre la recherche du début.
- Le fait d'utiliser des heuristiques stochastiques fait en sorte que la fouille visitera, avec une forte probabilité, de nouvelles parties de l'arbre de recherche à chaque redémarrage.

Les redémarrages

- Si on ne fait pas attention, une fouille avec redémarrage peut explorer plusieurs fois un même noeud et peut, avec de la malchance, ne jamais visiter certaines portions de l'arbre.
- Visiter deux fois un même noeud est un risque que l'on peut prendre. C'est inefficace, mais si le gain en diversité produit par les redémarrages est suffisant, la recherche s'en trouvera tout de même accélérée.
- L'omission d'explorer des noeuds de l'arbre de recherche n'est pas acceptable. Un de ces noeuds pourrait être la solution recherchée.

Stratégies de redémarrage

- Une stratégie de redémarrage est une séquence infinie $t_1, t_2, t_3, \ldots$ où t_i est un entier non négatif ou infini.
- Le solveur procède par itération. À la $i^{ème}$ itération, il explore l'arbre de recherche, arrête après t_i retours arrière et procède à un redémarrage.
- Si une des valeurs t_i est plus grande que le nombre de feuilles dans l'arbre de recherche, alors la stratégie de redémarrage garantit que toutes les solutions seront explorées à l'itération i.

Stratégies de redémarrage

- Parmi les stratégies de redémarrage fréquemment utilisées, on retrouve la série géométrique 1, r, r^2, r^3, r^4, ... pour une constante r > 1.
- La séquence de Luby 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, ... génère une recherche dont le nombre de retours arrière est à un facteur logarithmique de la séquence optimale (qui elle, est inconnue).
- Ces deux séries ont une tendance croissante et finiront par produire des nombres qui surpasseront le nombre de feuilles dans l'arbre de recherche. Elles garantissent donc que la totalité de l'arbre de recherche sera explorée.

Conclusions

- Il existe une multitude de façons de rendre un solveur plus robuste.
- Plusieurs heuristiques ont été mises au point afin d'accélérer la recherche d'une large variété de problèmes. Parmi les heuristiques de choix de variables on compte:
  - l'ordre statique, le plus petit domaine, la variable la plus contrainte, les heuristiques structurelles, les combinaisons d'heuristiques, les heuristiques stochastiques et les heuristiques adaptatives,

Conclusions

- Nous avons vu deux techniques qui diversifient la recherche et explorent différentes portions de l'arbre de recherche.
  - Fouilles basées sur les déviations telles que LDS.
  - Les redémarrages

Références

- Heuristiques
  - HCP: Chapter "Backtracking Search Algorithms" Section "Heuristics for Backtracking Algorithms".
- LDS
  - W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence*, pages 607-613, Montréal, 1995.