POLYTECH NANTES

École d'ingénieurs de l'université de Nantes

2019–2020

Spécialité « Informatique »

Introduction aux techniques de base de l'Intelligence artificielle

Points clés & guide de travail personnel

José MARTINEZ

— 4e année —

UNIVERSITÉ DE NANTES

POLYTECH

Premier réseau national
des écoles d'ingénieurs
polytechniques des universités

Site de la Chantrerie
Rue Christian Pauc - BP 5060
44306 Nantes cedex 3
France
tél. +33 (0)2 40 68 32 00
fax. +33 (0)2 40 68 32 32

Site de Gavy
Gavy Océanis - BP 152
44603 Saint-Nazaire cedex
France
tél. +33 (0)2 40 90 50 30
fax. +33 (0)2 40 90 50 24

www.polytech.univ-nantes.fr

Table des matières

1 Introduction

1.1 Diverses définitions et approches

Définition 1.

« L'IA est le domaine qui étudie comment faire faire à l'ordinateur des tâches pour lesquelles l'homme est encore le meilleur. » (RICH)

Définition 2 (Opérationnelle)**.**

« L'IA cherche à concevoir des programmes et des machines en mesure de traiter des problèmes pour lesquels on ne connaît pas encore de méthodes de résolution directes et assurées. » (LINDSAY)

Exercices.

— Test de TURING
— Objection de la « chambre chinoise »

1.2 Pseudo-formalisation

Définition 3 (Problème d'IA)**.**

Spécifiable via

1. *situation initiale*
2. *situations finales*
3. *transformations*

Corollaire 4.

1. *pas d'algorithme direct* pour trouver une solution
2. notion de *choix* (quelles transformations appliquer ? dans quel ordre ?), composante essentielle de l'intelligence

Corollaire 5.

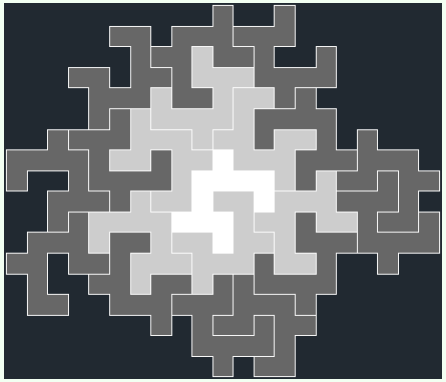
Beaucoup de problèmes d'IA se ramènent à des problèmes de *recherche / exploration* dans un graphe de taille *exponentielle*, voire infinie...

Exercices.

— « Sortir du cadre »
— Inversion des cavaliers
— Problème du zèbre

Exemple 1 (Espace de recherche infini)**.**

Le pavage du plan, et plus généralement les problèmes de tassement, génèrent des graphes de recherche infinis.
Par exemple, soit un ensemble de figures de type polyominos comme dans le jeu Tétris™.
Le but est de rechercher le plus petit assemblage de tels polyominos qui soit sans trou, sans symétrie et qui permette – utilisé comme élément de « base » – de paver le plan.



RHOADS, Glenn Charles. *Planar tilings and the search for an aperiodic prototile*. Thèse de Doctorat, Rutgers University, 2003.

2 Espaces de recherche

Définition 6 (Espace de recherche)**.**

Hyper-graphe
- ensemble de descriptions instantanées d'une situation quelconque
- ensemble d'(hyper-)arcs représentent des relations de filiation entre les nœuds

2.1 Graphes d'états

Définition 7 (Graphe d'états – naïf)**.**

— *nœud* = *état instantané* (l'ensemble des variables servant à décrire *et* résoudre un problème)
— *arc* = une *transformation* de l'état d'origine en l'état de destination (un groupe d'« affectations » ayant modifié tout ou partie de la mémoire)

Définition 8 (Graphe d'états – formalisé)**.**

Un *problème* Q est un sextuplet $\langle E, i, F, O, P, C \rangle$

— E un ensemble d'états
— $i \in E$ un état initial
— $F \subseteq E$ un sous-ensemble d'états finaux
— $O = \{o : E \nrightarrow E\}$ un ensemble d'opérateurs de transformation
— $P = \{p : O \times E \to \mathcal{B}\}$ un ensemble de prédicats associés définissant le cas d'applicabilité d'un opérateur sur un état
— $C = \{c : O \times E \to \mathbb{R}^+\}$ un ensemble de fonctions de coût associées

Remarque. Une instance d'opération o incorpore tous les paramètres complémentaires au seul état e.

Remarque. C peut ne pas être utilisé dans certains cas : $\forall (o, e), c(o, e) = 1$.

Exercices.

— Décrire le jeu des échecs avec le sextuplet $\langle E, i, F, O, P, C \rangle$.
— Est-on loin d'un programme de jeu d'échecs ? [a]

[a] Oui, il s'agit de trouver les « bonnes » parties parmi environ 10^{123} états… Reste applicable pour les problèmes du type « Faire mat en X coups ».

2.2 Recherche dans un graphe d'états

Définition 9 (Espace de recherche exploré)**.**

Un espace de recherche exploré de Q est un graphe orienté valué $G = (E', D, v)$ tel que
- $i \in E' \subseteq E$
- $D \subseteq \{(e_1, e_2) \in E' \times E' : \exists o \in O, p(o, e_1) \wedge e_2 = o(e_1)\}$
- G est connexe, c'est-à-dire $\forall e \in E', \exists (i = e_0, e_1, \ldots, e_k, \ldots, e_n = e)$ un chemin dans G

Définition 10 (Solution dans un graphe d'états)**.**

Une *solution* de Q est un *chemin* dans G entre l'état initial i et un état final $e \in F \cap E'$

$$s = (i = e_0, e_1, \ldots, e_n = e)$$

Définition 11 (Meilleure solution dans un graphe d'états)**.**

Une *meilleure solution* de Q est une solution de coût minimal, avec

$$\text{coût}(s) = \sum_{k=0}^{n-1} c(op(e_k, e_{k+1}), e_k)$$

2.3 Graphes de sous-problèmes

Définition 12 (Graphe de sous-problèmes – naïf)**.**

— *nœud* = *problème à résoudre / résolu* (paramètres et éventuellement valeur de retour)
— *arc* = la *décomposition* du problème en *un ensemble de sous-problèmes conjoints* (« appels » de fonctions)

Définition 13 (Graphe de sous-problèmes)**.**

Un *problème* Q est un sextuplet $\langle E, i, F, O, P, C \rangle$

— E un ensemble de ***problèmes***
— $i \in E$ un ***problème*** initial
— $F \subseteq E$ un sous-ensemble de ***problèmes résolvables « directement »***
— $O = \{o : E \nrightarrow \mathcal{P}(E)\}$ un ensemble d'opérateurs de ***réduction***
— $P = \{p : O \times E \to \mathcal{B}\}$ un ensemble de prédicats
— $C = \{c : O \times E \to \mathbb{R}^+\}$ un ensemble de fonctions de coût

Exercices.

— Définir les concepts d'« espace de recherche exploré », de « solution » et de « meilleure solution » dans un graphe de sous-problèmes.

2.4 Graphes d'états *vs* graphes de sous-problèmes

Théorème 14 (Équivalence)**.**

Tout problème peut être traduit aussi bien par un graphe d'états que par un graphe de sous-problèmes.

2.5 Modélisation

La modélisation, c'est-à-dire E, est ici laissée de côté.
On utilisera une approche « quelconque » : UML, relationnel, mathématique...

Remarque. Les approches dédiées à l'IA ont principalement introduit et rendu facile d'utilisation des concepts :

— de catégorisation (ancêtres de l'approche à objets avec des extensions comme les prototypes, la migration, les réflexes, des contraintes dynamiques) ;
— d'approximations / inconnues (valeurs nulles, floues, ensemblistes, d'intervalle). [1]

Tous ces aspects-là, s'ils sont nécessaires, doivent être explicitement exprimés dans une modélisation standard.

Remarque. L'approche relationnelle est à utiliser, sinon à privilégier, pour (i) sa capacité à modéliser sans anomalie et (ii) sa proximité avec Prolog.

1. Cet aspect reste particulièrement présent dans les solveurs de contraintes.

3 Techniques exploratoires aveugles

Fait 15.

Recherche et *non* choix ⇒ Utiliser la puissance de calcul *brute* de l'ordinateur

3.1 Explorations récursives dans un graphe d'états

Fait 16.

L'approche récursive amène à une exploration d'*arbre*.

Lemme 17.

La taille d'un arbre d'exploration peut être exponentielle en celle du graphe d'états sous-jacent, voire infinie...

Remarque. Le graphe peut déjà être de taille exponentielle, voire infinie, en lui-même...

3.1.1 Retours en arrière

A priori suffisant pour rechercher une (la première) solution quand l'espace de recherche est un arbre (sous-arbres *strictement* inclus) de taille finie...

Exemple 2 (Animation du Sudoku).

http://en.wikipedia.org/wiki/File:Sudoku_solved_by_bactracking.gif

Définition 18 (Recherche par retours en arrière).

$$R : \begin{array}{ccc} E & \not\to & O^{\mathbb{N}} \\ e & \mapsto & \begin{cases} () & \text{si} & e \in F \\ S & \text{si} & \exists o \in O, p(o,e) \wedge SR \neq \bot \\ \bot & \text{sinon} & \end{cases} \\ & & \text{avec } SR = R(o(e)), S = (o) \otimes SR \end{array}$$

Notations :
— $A^{\mathbb{N}}$ (ou A^*) dénote les séquences de taille variable d'éléments de A : $\bigcup_{i=0}^{+\infty} A^i$;
— $\bot$ dénote l'indétermination (ex. : NULL en SQL) ;
— $\otimes : A^{\mathbb{N}} \times A^{\mathbb{N}} \to A^{\mathbb{N}}$ dénote la concaténation de séquences (chaînes de caractères, listes, n-uplets).

Remarque. On peut renvoyer la séquence des états successifs au lieu des opérations, ou le couple opération et état résultant...

Codes (Le problème des grenouilles).

— *frogs_problem.pl*
— *frogs_solution.pl*
— *solver_naive.pl*
— *solver_naive_with_tracing.pl*
— *solver_common.pl*

3.1.2 Détection des circuits

Mémorisation des états *ancêtres* (A) de l'état courant (sous-graphe *moins* A strictement inclus dans graphe).

Définition 19 (Recherche par retours en arrière avec détection des circuits)**.**

$$
\begin{array}{rcl}
E \times \mathcal{P}(E) & \not\to & O^{\mathbb{N}} \\
R : \ (e, A = \emptyset) & \mapsto & \left\{ \begin{array}{lll} () & \text{si} & e \in F \\ S & \text{si} & e \notin A \wedge \exists o \in O, p(o,e) \wedge SR \neq \bot \\ \bot & \text{sinon} & \end{array} \right. \\
 & & \text{avec } SR = R(o(e), A \cup \{e\}), S = (o) \otimes SR
\end{array}
$$

Codes (Le problème des cruches)**.**

— *jugs_problem.pl*
— *jugs_solution_directed_cycle_detection.pl*
— *solver_directed_cycle_detection.pl*
— *solver_directed_cycle_detection_with_tracing.pl*

3.1.3 Recherche de profondeur bornée

Pour éviter les branches infinies : circuits dans le graphe d'états mais également espaces véritablement infinis et stériles... (strictement inclus mais non strictement décroissant)

Définition 20 (Recherche de profondeur bornée)**.**

$$
\begin{array}{rcl}
E \times \mathbb{N} & \not\to & O^{\mathbb{N}} \\
R : \ (e, B) & \mapsto & \begin{cases} () & \text{si} \quad e \in F \\ S & \text{si} \quad B > 0 \wedge \exists o \in O, p(o,e) \wedge SR \neq \bot \\ \bot & \text{sinon} \end{cases} \\
 & & \text{avec } SR = R(o(e), B-1), S = (o) \otimes SR
\end{array}
$$

Codes (Le problèmes des cruches, suite)**.**

— *jugs_solution_depth_limit.pl*
— *solver_depth_limit.pl*
— *solver_depth_limit_with_tracing.pl*

Exercices.

— Combiner la détection de circuits avec la recherche de profondeur bornée.

3.1.4 Recherche étagée

Pour rechercher *une meilleure* solution, *en nombre d'étapes*, en évitant l'écueil de la profondeur à choisir / trouver manuellement.

Définition 21 (Recherche étagée)**.**

$$
\begin{array}{rcl}
E \times \mathbb{N} \times \mathbb{N} & \not\to & O^{\mathbb{N}} \\
RE : \ (e, B = 0, B_{\max} = \infty) & \mapsto & \left\{ \begin{array}{lll} S & \text{si} & B \leq B_{\max} \wedge S \neq \bot \\ RE(e, B+1, B_{\max}) & \text{si} & B \leq B_{\max} \\ \bot & \text{sinon} & \end{array} \right. \\
 & & \text{avec } S = R(e, B)
\end{array}
$$

Théorème 22 (Complexité d'une recherche étagée)**.**

Le temps d'une recherche étagée n'est que deux fois plus important que celui de la recherche bornée avec la profondeur optimale.

Théorème 23.

Une recherche étagée est *similaire* à une recherche en *largeur d'abord*.

Corollaire 24 (Complétude de la recherche étagée)**.**

La recherche étagée est complète.

Codes (Le problème des cruches, encore).

— *jugs_solution_iterated_deepening.pl*
— *solver_iterated_deepening.pl*
— *solver_iterated_deepening_with_tracing.pl*

Remarque. La recherche étagée est en fait un « *méta-algorithme* » ; il peut prendre en paramètre n'importe quel algorithme R qui utilise une borne.

Remarque. La borne optimale pourrait être recherchée autrement que par une simple incrémentation : suite géométrique au lieu d'arithmétique, prise en compte de pas propres au problème, décrémentations depuis un maximum, recherches dichotomiques...

Exercices.

— Modifier l'algorithme pour utiliser les coûts associés aux opérations et fournir donc une recherche de coût borné.
— Quelle est la principale difficulté par rapport au théorème 22 ?

3.1.5 « *Memoing* » ou la détection des cycles

Pour éviter une *explosion combinatoire* due aux *cycles* dans les *graphes denses*.

Définition 25 (« *Memoing* » avec « effet de bord »)**.**

$$R : \begin{array}{ccc} E & \not\to & O^{\mathbb{N}} \\ e & \mapsto & \begin{cases} () & \text{si} & e \in F \\ M(e) & \text{si} & M(e) \neq \bot \\ S & \text{si} & \exists o \in O, p(o,e) \wedge SR \neq \bot \text{ avec } M \leftarrow M \cup \{(e,S)\} \\ \bot & \text{sinon} & \end{cases} \end{array}$$

avec $SR = R(o(e)), S = (o) \otimes SR$

avec $M \leftarrow \emptyset$ initialement

M mémorise ici les états déjà rencontrées – et résolus –, c'est-à-dire *certains* risques de cycles.
M peut être vu comme une généralisation et une *extension* progressive de F.

Remarque. Dans les cas où F est définie en extension, on pourrait initialiser M avec $\{(e, ()) : e \in F\})$ et retirer le cas $e \in F$ de R.

Dans le cas général, c'est un *cache*.

Remarque. De nombreuses stratégies existent pour gérer M de manière à éviter de stocker toute la partie du graphe de recherche déjà parcouru.

Exercices.

— Combiner avec la détection des circuits.

Remarque. Avec le non-déterminisme de Prolog, *plusieurs* solutions peuvent être trouvées pour un même état.
Il convient alors de n'en conserver qu'une, de préférence la moins longue (en tout cas pas toutes dans le cas où leur nombre est infini !).

Codes (Le problème des cruches, toujours).

— *jugs_solution_memoing.pl*
— *solver_memoing.pl*
— *solver_memoing_with_tracing.pl*

Remarque. Il s'agit encore d'un « *méta-algorithme* » ; n'importe quelle fonction utilisant des décompositions avec des intersections non vides peut bénéficier de cette technique.

Remarque. Il s'agit d'un échange algorithmique classique et gagnant : espace (polynomial) contre temps (exponentiel).

Remarque. Un cas extrême de « *memoing* » est la *programmation dynamique*.

Exercices.

— Ajouter une borne sur la profondeur maximale et utiliser avec la recherche étagée, en n'initialisant la mémoire qu'au début de cette dernière.

3.1.6 Séparation et évaluation

Pour rechercher *une* des *meilleures* solutions, en coût total des opérations.

Définition 26 (Recherche par séparation et évaluation – approche ascendante)**.**

$$R : \begin{array}{ccc} E \times \mathbb{R}^+ & \nrightarrow & O^{\mathbb{N}} \\ (e, B = \infty) & \mapsto & \begin{cases} () & \text{si} \quad e \in F \\ T(e, B, \{o \in O : p(o,e) \wedge c(o,e) \leq B\}) & \text{sinon} \end{cases} \end{array}$$

avec :

$$T : \begin{array}{ccc} E \times \mathbb{R}^+ \times \mathcal{P}(O) & \nrightarrow & O^{\mathbb{N}} \\ (e, B, A) & \mapsto & \begin{cases} \bot & \text{si} \quad A = \emptyset \\ T(e, B, A') & \text{si} \quad SR = \bot \\ S & \text{si} \quad D = \bot \\ D & \text{sinon} \end{cases} \end{array}$$

où :

- $A = \{o\} \cup A'$
- $SR = R(o(e), B - c(o,e), \{o' \in A' : c(o', o(e)) \leq B - c(o,e)\})$
- $S = (o) \otimes SR$
- $D = T(e, \text{coût}(S), A')$

Remarque. Si $B \neq +\infty \wedge \forall (c, o, e), c(o,e) = 1$, il s'agit d'une recherche de profondeur bornée.

Remarque. Si $B = +\infty$, risque de plongée dans une branche infinie.

Remarque. En général, les coûts ne doivent pas être nuls, sinon risque de plongée dans une branche infinie.

Codes (Le problème des cruches, fin)**.**

— *jugs_solution_uninformed_branch_and_bound.pl*
— *solver_uninformed_branch_and_bound.pl*
— *solver_uninformed_branch_and_bound_with_tracing.pl*

Exercices.

— Sur le même principe de la fonction T qui transfère de l'information entre branches explorées, écrire une recherche avec « memoing » mais sans « effet de bord »
— Combiner cette dernière avec l'approche par « séparation et évaluation ».
— Modifier la définition 26 pour tenir compte de la détection des circuits. [a]

a. Elle est déjà intégrée au solveur Prolog !

3.1.7 Comparaison

Définition 27 (Complétude)**.**

Un algorithme est complet s'il trouve une solution quand elle existe.

Définition 28 (Optimalité)**.**

Un algorithme est optimal s'il trouve la meilleure solution, quand elle existe.

		temps			
	complet	minimal	maximal	espace	optimal
retours en arrière	non, en général	$\Omega(d)$	$O(b^{d_{\max}})$	$O(d)$	non
retours en arrière avec détection de circuits	oui si espace fini	$\Omega(d)$	$O(b^{d_{\max}})$	$O(d)$	non
recherche avec « *memoing* »	oui si espace fini	$\Omega(d.\log\|M\|)$	$O(b^{d_{\max}}.\log\|M\|)$	$O(\|M\|)$	non
recherche étagée	oui	$2.O(b^d)$	$2.O(b^{d_{\max}})$	$O(d)$	oui si coûts unitaires
séparation et évaluation	oui si espace fini ou borne initiale finie	$\Omega(d)$	$O(b^{d_{\max}})$	$O(d)$	oui si borne initiale majorante

TABLE 1 – Comparaison entre algorithmes (avec d profondeur de la solution)

Exercices.

— À quoi correspondrait la recherche étagée avec « *memoing* » ?

Remarque. Tous ces algorithmes sont des « moteurs d'inférence » à « chaînage avant ».
Couplés à la modélisation des données de la définition 8, ils constituent des « systèmes experts ».
Prolog en lui-même est un système expert !

3.2 Explorations récursives dans un graphe sous-problèmes

Les algorithmes de la section 3.1 s'appuient sur la modélisation de la définition 8.

Tous ces algorithmes ont leur pendant dans les recherches en graphes de sous-problèmes (cf. définition 13)… mais ils sont plus difficiles à décrire.

Exercices.

— Transposer les algorithmes de la section 3.1 à des graphes de sous-problèmes !

Toutefois, la résolution *directe* d'un problème par une approche récursive génère *implicitement* – le plus souvent – un graphe de sous-problèmes.

Exemple 3.

Résoudre le problème du tri par l'algorithme du tri rapide de C. A. R. HOARE consiste à décomposer le problème en deux sous-problèmes *distincts* dont les *deux* sous-solutions *combinées* sont la solution au problème d'origine.

Exercices.

— Tours de Hanoï sans « effet de bord »
— Ali BABA et les quarante voleurs

Un graphe de sous-problèmes peut présenter les mêmes difficultés qu'un graphe d'états : circuits, cycles, sous-graphes stériles, etc., ce qui limite parfois le recours à une telle résolution directe.

Exemple 4.

Ne jamais programmer le calcul des combinaisons avec la définition récursive suivante, où $(n, p) \in \mathbb{N}^2$ avec $0 \leq p \leq n$:

$$\begin{cases} C_n^0 &= 1 \\ C_n^n &= 1 \\ C_n^p &= C_{n-1}^p + C_{n-1}^{p-1} \end{cases}$$

Pourquoi ? [a]

a. Tracer le graphe de calcul sur le triangle de Pascal. Le triangle de Pascal est un exemple de programmation dynamique (cf. remarque 3.1.5).

4 Heuristiques et recherches informées

Fait 29.

Recherche *et* choix ⇒ Utiliser *raisonnablement* la puissance de calcul de l'ordinateur

Fait 30.

Il s'agit ici de fournir au système une connaissance *experte* sur le problème, c'est-à-dire d'introduire enfin un peu d'« intelligence » dans le programme.

4.1 Familles d'approches

1. algorithme *exact* dans les *cas pratiques*
2. algorithme *efficace* pour des *sous-classes* de problèmes
3. algorithme *approximatifs*, avec garanties...
4. algorithme *approximatifs*, sans garanties...

4.2 Non-déterminisme ou le choix parfait

Définition 31 (Oracle)**.**

Un oracle, *choix_nd(X)*, renvoie toujours la (une) valeur $x \in X$ qui amène à une solution, si elle existe (échec implicite, $\perp$, si $X = \emptyset$).

Définition 32 (Recherche *sans* retours en arrière avec oracle)**.**

$$\begin{array}{rcl} E & \not\to & O^{\mathbb{N}} \\ R: \quad e & \mapsto & \begin{cases} () & \text{si} \quad e \in F \\ S & \text{si} \quad SR \neq \perp \\ \perp & \text{sinon} \end{cases} \end{array}$$

avec $o = \text{choix\_nd}(\{o \in O : p(o, e)\}), SR = R(o(e)), S = (o) \otimes SR$

Exercices.

— Résoudre le problème du voyageur de commerce avec un oracle.

En pratique, l'oracle est implémenté par :

1. un choix *le plus informé possible* ;
2. des retours en arrière, quand mauvais choix antérieur.

Remarque. Dans le langage Prolog, il s'agit tout simplement de `member(X, L)` !

4.3 Heuristiques pour recherches totales

Choix le plus informé possible ⇒ ordonnancement, voire sélection (seules branches possibles, voire meilleures branches *a priori* d'abord)

4.3.1 Détection des impasses

Définition 33 (Impasse)**.**

Un état $e \in E$ est une impasse si, et seulement si, quels que soient les chemins issus de e dans le graphe de recherche, aucun n'arrive à un état final $f \in F$.

Remarque. Une impasse *n'est pas* un état sans opération applicable.

Exercices.

— Récrire les définitions 18, 19, 20, 25 et 26 en y ajoutant le cas de détection d'une impasse.[a]

a. Mettre « $\top$ si Impasse(e) » en 2e cas.

Exercices.

— Sous quelle condition algorithmique est-il intéressant de fournir un prédicat d'évaluation d'une impasse ? [a]
— Quantifier le gain sur une recherche dans un *arbre énaire* en posant vos hypothèses de travail. [b]

a. Soit des opérations de base en $O(1)$. Soit des états qui génèrent en moyenne n fils. La recherche récursive complète *sans* détection d'impasse est alors en $O(n^p)$ où p est la profondeur de recherche moyenne.
Soit une impasse évaluée en $O(m)$. Supposons que la détection d'impasse réussisse à ne conserver que $0 < f < n$ fils en moyenne, répartis de manière équiprobable sur l'ensemble des états. La recherche récursive est alors en $O(m.f^p)$.
Pour fixer les idées, si la recherche complète développe un arbre ternaire et que l'impasse élimine un fils sur trois, alors on passe à l'exploration d'un arbre binaire. Ce gain « exponentiel » est d'autant plus gagnant qu'il est payé par un facteur m plus petit.
Bien entendu, ce n'est pas la seule situation qui peut se présenter…

b. La recherche dans un sous-graphe est généralement exponentielle. Il suffit que la détermination de l'impasse soit seulement polynomiale pour être largement gagnant.

Remarque. Une impasse est proche de la négation de la *précondition*.

Elle n'est pas incluse dans les préconditions des opérations car ces dernières contrôlent seulement l'applicabilité de l'opération, pas son effet global sur un état.

Par ailleurs, elle devrait alors être répétée dans les préconditions de toutes les opérations.

De plus, elle peut n'être que partielle, la véritable précondition de la fonction devant décrire tous les cas où une solution existe.

Or, par définition partielle de la fonction, nous admettons ne pas savoir si la solution existe ou pas.

4.3.2 Ordonnancement des choix

Il est préférable qu'une ou une meilleure solution se trouve dans le premier fils d'un état.
En cas de chance extrême, elle serait même trouvée en un temps linéaire !

Définition 34 (Recherche par séparation et évaluation – heuristique d'ordonnancement)**.**

$$R : \begin{array}{ccc} E \times \mathbb{R}^+ & \not\to & O^{\mathbb{N}} \\ (e, B = \infty) & \mapsto & \begin{cases} () & \text{si} \quad e \in F \\ T(e, B, \textit{tri\_heuristique}(\{o \in O : p(o,e) \wedge c(o,e) \leq B\})) & \text{sinon} \end{cases} \end{array}$$

Remarque. La version suivante, avec heuristique, se ramène à celle-ci avec une simple fonction d'évaluation uniformément... nulle !

Codes (Le problème des cruches, retour)**.**

— *jugs_solution_informed_branch_and_bound.pl*

4.3.3 Évaluation heuristique

Définition 35 (Fonction d'évaluation de coût heuristique *minorante*)**.**

Soit $H = \{h : O \times E \to \mathbb{R}^+\}$ tel que $\forall (h, o, e), h(o, e) \leq \text{coût}(R_{BB}(e))$.

Définition 36 (Recherche par séparation et évaluation – heuristique d'évaluation minorante)**.**

$$R : \begin{array}{ccc} E \times \mathbb{R}^+ & \nrightarrow & O^{\mathbb{N}} \\ (e, B = \infty) & \mapsto & \begin{cases} () & \text{si} \quad e \in F \\ T(e, B, tri(\{o \in O : p(o, e) \wedge h(o, e) \leq B\}), \lambda o.h(o, e)) & \text{sinon} \end{cases} \end{array}$$

avec :

$$T : \begin{array}{ccc} E \times \mathbb{R}^+ \times \mathcal{P}(O) & \nrightarrow & O^{\mathbb{N}} \\ (e, B, A) & \mapsto & \begin{cases} \bot & \text{si} \quad A = \emptyset \\ T(e, B, A') & \text{si} \quad h(o, e) > B \\ T(e, B, A') & \text{si} \quad SR = \bot \\ S & \text{si} \quad D = \bot \\ D & \text{sinon} \end{cases} \end{array}$$

où :

- $A = \{o\} \cup A'$
- $SR = R(o(e), B - c(o, e), \{o' \in A' : c(o', o(e)) \leq B - c(o, e)\})$
- $S = (o) \otimes SR$
- $D = T(e, \text{coût}(S), A')$

Remarque. Le test supplémentaire dans la fonction T est nécessaire car la borne B peut décroître, éliminant donc des branches qui étaient auparavant ouvertes.

Remarque. Une impasse (cf. définition 33) peut également être vue comme une heuristique d'évaluation de coût infini.

Codes (Le taquin)**.**

— *puzzle8_problem.pl*
— *puzzle8_solution.pl*
— *puzzle15_problem.pl*
— *puzzle15_solution.pl*
— *solver_informed_branch_and_bound.pl*
— *solver_informed_branch_and_bound_with_tracing.pl*

4.3.4 Propagation de contraintes

Fait 37.

Imposer des contraintes diminue la taille de l'espace de recherche, aussi bien (i) statiquement (E) que (ii) dynamiquement (R).

Fait 38 (Principes de la propagation de contraintes).

1. Choisir un des éléments les plus contraints.
2. Propager les conséquences du choix (au maximum).

Remarque. Quand un choix est forcé (élément présentant une unique possibilité), alors l'exploration laisse place à la résolution (partielle) !

Exemple 5 (Les n reines).

— E
  1. représentation directe (matrice $n \times n$) : $A^8_{8\times 8} \approx 10^{14}$
  2. représentation par colonne (tableau n) : $8!.8^8 \approx 10^{11}$

— R
  1. pose ordonnée : $8^8 \approx 10^7$ (cf. http://www.youtube.com/watch?v=ckC2hFdLff0)
  2. ligne choisie dans un ensemble : $8! \approx 10^4$
  3. propagation de contraintes : $\approx 10^?$

Exercices.

— Taille d'un espace de recherche

4.3.5 Similitudes

Définition 39 (Prédicat de la similitude entre états)**.**

Soit $\precsim : E \times E \to \mathcal{B}$, un prédicat traduisant une relation de pré-ordre (réflexivité et transitivité) partiel ou total.

Remarque. L'égalité est une similitude très forte.

Définition 40 (« *Memoing* » étendu à la similarité – avec « effet de bord »)**.**

$$R : \begin{array}{ccc} E & \not\to & O^{\mathbb{N}} \\ e & \mapsto & \begin{cases} () & \text{si} & e \in F \\ \bot & \text{si} & \exists e' \precsim e, M(e') \neq \bot \\ S & \text{si} & \exists o \in O, p(o,e) \wedge SR \neq \bot \text{ avec } M \leftarrow M \cup \{(e,S)\} \\ \bot & \text{sinon} & \end{cases} \\ & & \text{avec } SR = R(o(e)), S = (o) \otimes SR \end{array}$$

avec $M \leftarrow \emptyset$ initialement

Remarque. Le cas où la similitude est détectée pourrait renvoyer également l'indétermination dans l'algorithme 25.

4.4 Heuristiques pour recherches partielles / parfaites

Fait 41.

Un espace de recherche, même exploré « intelligemment », peut encore se révéler trop vaste.

4.4.1 Impasse heuristique

Définition 42 (Impasse heuristique)**.**

Un état $e \in E$ est une impasse heuristique si, et seulement si, *peu* de chemins issus de e dans le graphe de recherche arrivent à un état final $f \in F$.

Remarque. Une impasse heuristique peut ne plus permettre de trouver de solution.

Inversement, on peut démontrer que les états finaux non atteints par cet état le restent par d'autres états.

Etc.

Bref, la définition d'une impasse heuristique peut tout aussi bien être un théorème complexe sur le problème qu'une idée empirique.

4.4.2 Ordonnancement des choix et choix restreints

Seules quelques meilleures branches *a priori* sont poursuivies, voire une unique meilleure branche *a priori*.

Définition 43 (Recherche par séparation et évaluation – heuristique de sélection)**.**

$$R : \begin{array}{ccc} E \times \mathbb{R}^+ & \not\to & O^{\mathbb{N}} \\ (e, B = \infty) & \mapsto & \begin{cases} () & \text{si} \quad e \in F \\ T(e, B, \textbf{sélection\_heuristique}(\{o \in O : p(o,e) \wedge c(o,e) \leq B\})) & \text{sinon} \end{cases} \end{array}$$

Remarque. La sélection heuristique peut utiliser un tri heuristique en entrée ou en sortie.

Exercices.

— Proposer une variante où le nombre de fils retenus (où largeur) est variable en fonction des valeurs estimées des fils.
— Proposer une variante où le nombre de fils retenus (où largeur) est fixe mais peut être augmenté progressivement comme dans la recherche étagée.

Remarque. Si la sélection heuristique se résume à un seul – *a priori* meilleur – élément alors *descente de gradient* (si critère numérique à optimiser) ou algorithme « glouton ». Dans les deux cas, optimum *local* si espace non convexe.

Remarque. Véritable algorithme glouton si l'heuristique est *parfaite*, mais il n'y a plus vraiment de recherche.

Définition 44 (Système commutatif)**.**

Certains choix peuvent être faits dans un ordre quelconque, ils amènent tous aux mêmes solutions. Un choix non-déterministe devient alors un choix quelconque.
Un système est dit commutatif s'il vérifie les conditions suivantes :

1. $\forall e \in E, \forall o \in O : p(o,e), R(e) \neq \perp \Rightarrow R(o(e)) \neq \perp$;
2. $\forall e \in E, \forall (o_1, o_2) \in O^2 : o_1 \neq o_2 \wedge p(o_1, e) \wedge p(o_2, e), p(o_2, o_1(e))$;
3. $\forall e \in E, \forall \pi, o_{\pi(1)} \circ o_{\pi(2)} \circ \ldots o_{\pi(i)} \circ \ldots o_{\pi(n-1)} \circ o_{\pi(n)}(e) = o_1 \circ o_2 \circ \ldots \circ o_i \circ \ldots \circ o_{n-1} \circ o_n(e)$.