IA ET ETERNITY

Tristan Cazenave

Laboratoire d'Intelligence Artificielle
Département Informatique, Université Paris 8,
2 rue de la Liberté, 93526 Saint Denis, France.

cazenave@ai.univ-paris8.fr

Résumé: Dans ce papier, nous exposons la façon dont le puzzle Eternity a été résolu par AlexSelby et Olivier Riordan. Nous mettons en perspective les méthodes qu'ils ont utilisé avec des techniques d'Intelligence Artificielle similaires.

Mots Clés: Recherche heuristique, puzzle, Eternity.

Introduction

Eternity est un puzzle. Il a été commercialisé en Angleterre en 1999 au prix de 30 Livres. Ce fut un des jeux qui y fut le plus vendus en 1999/2000 : le nombre de puzzles vendus dépassa le nombre des ventes de Trivial Pursuit. Il est vrai que son auteur, Christopher Monckton un ancien conseiller de Mrs Tatcher, avait offert un prix de 1 million de Livres à la première personne qui trouverait une solution. Ce n'est pas la peine de vous précipiter pour acheter le puzzle, il a été résolu en Mai 2000 par Alex Selby (70%), Dr en Mathématiques de Cambridge, 3ème Dan au jeu de Go, sans emploi, programmeur de Go potentiel... et Olivier Riordan (30%), Dr en Mathématiques de Cambridge, chercheur au Trinity College. Ils ont devancé de peu le grand maître d'Echecs par correspondance Guenter Stertenbrink, qui a trouvé une deuxième solution en Juillet 2000.
Nous allons décrire les méthodes qu'Alex Selby et Olivier Riordan ont employé pour résoudre ce problème, à partir de notes d'Alex Selby [Selby].

Les pièces et le puzzle

Eternity compte 209 pièces. Le but est de remplir complètement un dodécagone (forme régulière à douze cotés égaux) avec les 209 pièces qui sont des polydrafters. Les polydrafters sont des pièces composées de n triangles rectangles (30-60-90). Il y a 6 didrafters, 14 tri, 64 tetra, 237 penta, 1014 hexa, 4124 hepta et 17705 octadrafters. Aucune des pièces d'Eternity n'a d'angle de 30 degrés ou de triangles vides. Les polygones qui ont cette propriété s'appellent des polydudes. Avec ces contraintes, il y a 3 didrafters, 1 tridrafter, 9 tetradrafters, 15 pentadrafters, 59 hexadrafters, 152 heptadrafters, 517 octadrafters, 1547 nonadrafters, and 5064 decadrafters, 16123 hendecadrafters, and 52630 dodecadrafters.
Eternity utilise des dodécadudes (pièces composées de 12 triangles rectangles sans angle de 30 degrés ou de triangle vide).

La création du puzzle est facile, pas sa résolution

Contrairement à sa résolution, la création du puzzle est facile. On peut tout d'abord remarquer que toutes les pièces ont la même aire (6 triangles équilatéraux ou 12 triangles rectangles). De plus, il existe 770 pièces qui ont des propriétés convenables pour faire partie du puzzle. Il est assez facile de remplir une figure de taille 209 avec 770 pièces. Il suffit de choisir les pièces qui correspondent aux aires vides à la fin. En revanche, il est beaucoup plus dur de remplir un figure de taille 209 avec 209 pièces. A la fin, il reste très peu de choix pour les aires restées vides.

Régions, aires, états et sites.

On définit la taille d'une région comme l'aire de cette région divisée par l'aire d'une pièce. Un état est une région associée à un ensemble de pièces. S'il y a plus de pièces que la taille de la région, il restera des pièces lorsque la région sera remplie, et il peut y avoir plus d'une solution. Un site est un espace vide connexe sur une région.

La force brute

La solution la plus simple qui vient à l'esprit pour résoudre Eternity est la force brute : une recherche exhaustive de toutes les possibilités de remplir une région. Cela ne marche pas pour trouver la solution du puzzle complet, mais cela est utile pour résoudre des puzzles de tailles plus petites.
On peut utiliser l'algorithme de force brute suivant pour paver une région : Etant donné un mécanisme pour choisir un site, on considère une branche par façon de poser une pièce sur le site choisi. Aucune solution ne manque puisque le site doit être couvert pour toutes les solutions. Aucune solution n'est comptée deux fois puisque chaque pièce placée amène à un ensemble disjoint de solutions. L'algorithme marche toujours si on a plus de pièces que la taille de la région.
Cette méthode de remplissage est plus adaptée à Eternity que de choisir la prochaine pièce et de backtracker sur ses placements possibles. On peut avec cette méthode choisir le site à paver. Intuitivement, on voit bien qu'il faut choisir le site pour lequel on a le moins de façons possibles de placer une pièce. Si il y a une impossibilité dans la région, il faut la détecter le plus tôt possible, et ne pas perdre de temps à backtracker sur des régions faciles alors qu'il n'y a pas de solutions. Cette propriété d'Eternity se rencontre aussi dans d'autres problèmes. Par exemple pour le coloriage de cartes, il est inutile de continuer de colorier une partie facile de l carte si on est en présence d'une impossibilité sur une autre partie de la carte. On peut gagner beaucoup de temps en détectant rapidement les impossibilités. Le parallèle est aussi frappant avec l'algorithme de recherche avec partition de M. Ginsberg [Ginsberg 1996]. Son algorithme consiste à mémoriser des sous ensembles importants de positions déhà cherchée et à les réutiliser ensuite pour évaluer les nouvelles positions similaires. Il obtient ainsi des gains très importants pour l'Alpha-Béta sur la résolution de donnes ouvertes au Bridge. Un autre problème pour lequel de telles méthodes sont utiles est Sokoban [Junhganns and Schaeffer 2001], dans lequel on peut détecter à l'avance des "deadlocks" qui empêchent de trouver une solution.
L'heuristique de commencer par le choix le plus contraint, est aussi une heuristique bien connue des systèmes de résolution de contraintes [Lauriere 1978], [Gent et al. 1996]. Alex Selby propose de valider théoriquement cette intuition en faisant les hypothèses suivantes : le

facteur de branchement étant bp=f(p), p étant la profondeur, le nombre de nœuds de la recherche est égal à $N=1+b_0(1+b_1(1+b_2(1+...)))$. Pour un n et un r donnés, N peut se réécrire $N=b_n(K+b_{n+r}.L)$ avec K et L>0 dépendants des autres b_i. Si on peut faire varier b_n et b_{n+r} en gardant $b_n.b_{n+r}$ constant et les autres b_i inchangés (ce qui correspond a intervertir le choix de deux sites) alors, puisque $b_n(K+b_{n+r}.L)=K.b_n$+constante, pour minimiser N, on doit minimiser b_n (maximiser b_{n+r}). Ce raisonnement est bien conforme à l'heuristique qui consiste à choisir le site qui à le plus petit facteur de branchement en premier.
Le problème avec cette modélisation de la recherche est que l'arbre de recherche n'est pas aussi uniforme. Et que suivant le coup joué, les arbres de recherche suivants diffèrent beaucoup. De plus les sites d'Eternity ne sont pas vraiment indépendants. Par exemple, étant donné deux sites A et B. On a 3 façons de remplir A et 4 façons de remplir B. D'après l'heuristique précédente A est meilleur. Mais si après chaque remplissage de A on a encore plusieurs remplissages possibles alors qu'après 2 des façons de remplir B on est bloqué... alors B est peut être meilleur que A d'un facteur 2 ou 3 ! Il est tout de même coûteux d'avoir cette information puisqu'on doit examiner ce qui se passe après le pavage de A et de B. Toutefois c'est parfois utile de passer du temps à avoir une bonne approximation du "vrai" branchement (ça ne marche pas pour Eternity mais ça peut être utile pour d'autres problèmes...).
Il peut exister une meilleur façon de calculer le facteur de branchement bd que de faire une recherche à profondeur d et de compter les feuilles. C'est ce que nous allons voir maintenant.

Améliorations sur la force brute

L'idée consiste à associer à chaque pièce le nombre p de façons de paver chaque site (pour Eternity $p\approx0.03$). Si on note b_i, le nombre de façons de paver un site après avoir placé i pièces. Pour un puzzle de N pièces (sur un région de taille N) avec des pièces similaires on a $b_i =(N-i)p$. Le nombre de solutions du puzzle est le produit pour i allant 0 à N-1 soit $N!t^N$. Il existe N tel que $N!t^N > 1$, ce qui nous permet de calculer la taille critique du puzzle au dessus de laquelle il y a des solutions. Soit s la taille critique du puzzle. Pour Eternity $s \approx 75$. Or Eternity a 209 pièces ce qui est très supérieur à 75. Le nombre de solutions d'Eternity est proche de 10^{95} avec cette modélisation. On peut noter que pour N<s on a un puzzle sur-contraint. La meilleure méthode est alors la force brute. Alors que pour N>s le puzzle est sous contraint et on peut alors s'intéresser à des méthodes de type recherche locale ou métaheuristiques qui font une recherche plus ou moins aléatoire de l'espace de recherche.
Pour une taille N>s, le problème n'est pas plus dur que pour s, puisqu'on peut poser le N-s premières pièces et résoudre un problème de taille s. De plus on peut poser les N-s premières pour se mettre en position favorable. Reste a définir ce qu'est une position favorable. Ici on peut faire un analogie avec les problèmes de binpacking (remplisage d'un coffre de voiture par exemple) : on cherche à mettre en premier les formes bizarres , et à mettre les formes faciles à placer en dernier. De plus, à tout moment l'espace qui reste ne doit pas être trop fragmenté… (garder une bonne forme). Les mauvaises configurations auront beaucoup de trous alors que les bonnes configurations auront frontière convexe. L'algorithme de recherche consiste donc en deux phases. Une phase de remplissage qui tend à amener à des états favorables de taille relativement petite (≈ 30) aussi vite et bien que possible. Une seconde phase qui consiste à utiliser la force brute sur ces états. Pour des états suffisamment petits, il est payant de les explorer complètement. Le temps passé à les explorer est plus petit que le temps passé à les créer.
Pour Eternity un état favorable est :

- un état ayant de bonne pièces (facilement plaçables)
- un état ayant une belle frontière.

Une fois cette observation heuristique faite, il reste à la représenter avec des nombres. De plus, des améliorations portant sur des facteurs du second ordre comme par exemples les pièces qui marchent bien ensemble peuvent être ajoutées.

Estimation de la difficulté des pièces

Pour estimer la difficulté de trouver une solution avec une pièce, on peut trouver toutes les solutions à des problèmes de petite taille, et compter le nombre de fois où la pièce est présente dans les solutions. Reste à choisir la taille des petits problèmes à résoudre. Pour N=100, la recherche brute ne peut pas marcher car le puzzle est trop grand et impossible à explorer. Pour N=40, c'est encore trop difficile, de plus il n'y a pratiquement pas de solutions à cause de la taille critique d'Eternity. Il n'y a donc pas de taille vraiment utilisable avec cette méthode. On doit donc faire des approximations.
Pour approximer la difficulté des pièces, on peut faire un recherche exhaustive sur des états qui ont plus de pièces que la taille de la région. Ils auront beaucoup plus de solution, et cela donnera un estimation de la difficulté réelle de poser les différentes pièces. Par exemple, on peut essayer de paver des régions de taille 24 avec 90 pièces. La probabilité de pouvoir paver un région de taille 24 avec 24 pièces est de 10^{-17}. C'est donc en pratique impossible. En revanche, il y a $4x10^{21}$ façons de choisir 24 pièces parmi 90. On arrive donc à une résultat de 40000 solutions (en fait 40000/4=10000 solutions à cause de la parité). De plus le temps de recherche exhaustif pour placer 90 pièces sur une région de taille 24 est raisonnable (10 minutes pour des milliers de solutions). En répétant cette recherche des milliers de fois, on arrive en quelques semaines à approximer la difficulté des pièces.

Estimation de la difficulté des régions

Pour calculer la probabilité qu'une certaine région peut être remplie avec un certain ensemble de pièces, on utilise les probabilités calculées pour chaque pièce. La probabilité que la région soit pavée est le produit des probabilités des pièces pour un état (en fait A. Selby utilise les logarithmes des probabilités, ce qui lui permet de les additionner ce qui est mopins couteux que les multiplications). A ces probabilités associées aux pièces composant un état, on peut ajouter des paramètres mi sur la forme de la région. On a alors une évaluation de la probbilité de paver un état.

Algorithme de recherche

Il ne reste plus alors qu'à utiliser un algorithme de recherche qui va explorer en premier les états les plus prometteurs donné par l'évaluation des étts décrite aux sections précédentes. L'algorithme retenu par A. Selby est simple :

1) Décider du site le plus contraint
2) Engendrer tous les pavages de ce site
3) Prendre les n meilleurs (largeur=n)
4) Goto 1

La solution d'Eternity a été trouvée avec une largeur de 10 000 et un lookahead de 2.

Epilogue

Alex Selby et Olivier Riordan se sont donc partagé un million de Livres pour avoir été les premiers a avoir résolu Eternity. Il est possible que Christopher Monckton édite un deuxième puzzle de type Eternity, qui pourrait être doté d'un prix de 5 millions de £. Il est étonnant que la communauté IA ne se soit pas interéssée à ce problème qui rentre pourtant tout a fait dans son champ de recherche lorsque le premier puzzle Eternity est sorti. Il serait probablement intéressant de constituer un petit groupe de chercheurs en IA interéssés par le sujet si un autre puzzle de ce type est édité…

Bibliographie

[Gent et al. 1996] Ian P. Gent, Ewan MacIntyre, Patrick Prosser, Barbara M. Smith and Toby Walsh: An Empirical Study of Dynamic Variable Ordering Heuristics for the Constraint Satisfaction Problem. CP 96, LNCS. pp 179-193, 1996.

[Ginsberg 1996] Ginsberg M. L. *Partition Search*. Proceedings de AAAI96.

[Junghanns and Schaeffer 2001] A. Junghanns and J. Schaeffer. Sokoban: Enhancing General Single-Agent Search Methods Using Domain Knowledge, *Artificial Intelligence 129 (1-2)*: 219-251 (2001).

[Lauriere 1978] J.-L. Lauriere, *A Language and a Program for Stating and Solving Combinatorial Problems*. Artificial Intelligence 10 (1978) 29-127.

[Selby 2001] Selby A. Notes from a talk. http://www.archduke.demon.co.uk/eternity/talk/notes.html. January 2001.