

Les enjeux du choix du chemin

Même en implémentant les contraintes, il existe trop de combinaisons possibles pour que l'ordinateur puisse toutes les tester en un temps raisonnable. Il est donc nécessaire d'avoir recours aux probabilités.

En situation d'équiprobabilité, moins le nombre de possibilités est grand, plus la probabilité de tomber sur un élément est grande. Cette approche intuitive est formalisée par l'assertion suivante :

$$\forall x \in E, P(x) = \frac{1}{\text{card}(E)}$$

Plus les contraintes sont restrictives, plus le nombre de possibilités va être restreint, donc plus la probabilité de trouver la solution est grande.

Comme le nombre de combinaisons est trop grand, l'ordinateur ne reviendra probablement pas jusqu'aux premières pièces. Il est donc essentiel de se diriger dès le départ vers la ou une des bonnes branches, et cela grâce aux probabilités. Nous devons donc placer les pièces possédant le plus de contraintes en premier.

Le complexe contraintes/proximité

L'algorithme de backtracking présente cependant des limites : lorsqu'il s'aperçoit qu'une des pièces qu'il a posé antérieurement est invalide, il devra faire le chemin inverse pour la modifier. Le problème est que le temps pris en fonction de la distance des pièces n'est pas linéaire, mais dérivé de la factorielle :

$$t_{n+1} = k_{n+1} \times t_n$$

t_n étant le temps mis pour tester toutes les possibilités de n positions, et k_{n+1} étant le nombre de pièces à tester pour la $n+1$ ème position.

Il est donc important que l'on teste le plus rapidement possible la validité des pièces pour éviter d'avoir à parcourir un chemin trop long, or la validité des pièces se fait par la correspondance des motifs, donc par le test des pièces voisines. Ainsi, pour une optimisation de l'algorithme, il est nécessaire que si une pièce est incertaine, alors la position suivante à tester soit une pièce voisine.

Prenons un exemple pour être plus parlant : si nous commençons par positionner les 4 coins, puis nous commençons à remplir du haut vers le bas. En arrivant en bas, l'algorithme s'aperçoit que le coin en bas n'est pas le bon. Il lui faut tout remonter pour revenir à cette pièce, effectuant ainsi des milliards de tests inutiles.

La proximité des pièces n'est importante que si la pièce posée est incertaine. Elle ne s'applique donc pas à la pièce fixe. Ainsi, la seconde pièce posée pourra être en tout point du plateau.

Une fois la pièce fixe posée, nous en revenons à la première considération qui consiste à se demander quelle est la seconde position qui a le plus de contraintes. Nous voyons tout de suite que les coins sont les secondes positions ayant le plus de contraintes, car elles n'ont chacune que 4 pièces possibles. Une fois un coin posé, celui-ci étant incertain, l'obligation de proximité entre en jeu et oblige à poser une pièce parmi celles qui entoure le coin.

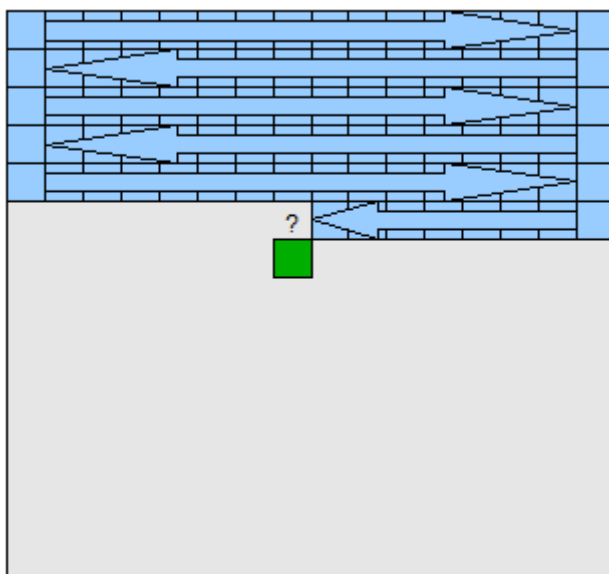
Ensuite, les contraintes que nous allons voir correspondent à un couple de 2 motifs. Prenons un exemple : si nous posons en seconde position le coin en haut à gauche, les positions voisines sont (2,1) et (1,2). Dans les 2 cas, des contraintes sur les motifs en haut et à gauche s'imposent. Ensuite, pour garder cette double contrainte, nous allons être obligés de longer les bords, puis revenir suivant le mouvement d'un serpent.

Est-il nécessaire de poser la pièce fixe en premier ?

Imaginons que nous ne posons pas la pièce fixe en premier. Nous commençons par un coin, suivons le mouvement d'un serpent. Au bout d'un certain nombre de pièces, nous arriverons à la position de la pièce fixe. Les règles d'éternity imposant la pièce 139 à cette position, il est nécessaire de rajouter une condition dans la fonction de backtracking pour forcer l'utilisation de la pièce imposée.

Les problèmes d'optimisation se font voir à 2 niveaux : une condition est testée très rapidement par un ordinateur, mais une condition effectuée des milliards de fois a un impact au niveau du temps global d'exécution de l'algorithme.

Le second problème d'optimisation se situe lors du choix de la pièce se positionnant au-dessus de la pièce fixe. Il est clair que celle-ci devrait avoir 3 contraintes : le motif supérieur, le motif à gauche ou à droite suivant le sens du mouvement, et le motif inférieur qui correspond au motif supérieur de la pièce fixe. Si la pièce fixe n'est pas déjà posée, cette troisième contrainte ne sera pas prise en compte, donc trop de pièces seront considérées comme valables et de nombreux tests inutiles seront effectués.



Revient-il au même de suivre un chemin en escargot autour de la pièce fixe ?

Appliquons les principes énoncés plus hauts : une fois posé la pièce fixe, comme celle-ci n'est pas incertaine, la règle de la proximité de s'applique pas. Nous avons donc totalement le choix quand au positionnement. En revanche, d'après la règle des contraintes les plus fortes en priorité, nous devons placer la seconde pièce au lieu où les contraintes seront les plus grandes. Quel que soit la position autour de la pièce fixe, la seule contrainte sera 1 motif de celle-ci. Les seuls endroits où les contraintes sont les plus fortes sont les coins, car ils appliquent des contraintes sur 2 motifs. Il est donc nécessaire de choisir un coin comme 2e position pour optimiser l'algorithme, or ensuite, la loi de proximité s'applique, donc le chemin en escargot n'est pas optimisé. **Choisir le chemin en escargot est donc une erreur d'optimisation.**

Conclusion

En conclusion, voici le chemin le plus optimisé pour l'algorithme de backtracking de résolution d'eternity II est le suivant :

