Aux petites mises, grands gains !

1 le background

Le problème « Mal à la Tetris » propose d'explorer tous les placements possibles de formes géométriques dans un rectangle quadrillé. Comme indiqué dans son énoncé, le nombre de placements à tester augmente exponentiellement avec la taille du rectangle et le nombre de formes. Par exemple, avec 10 formes, l'exécution d'une version très naïve dure environ 2.6 secondes pour un rectangle de 6×5, et plus de 19 secondes pour un 7×5 !

Pour réduire ce temps, il existe principalement deux techniques, qui sont souvent employées de concert : l'optimisation du code et l'utilisation d'heuristiques. Dans les deux cas, on cherche à réduire l'exploration des solutions. La différence se situe dans la façon de limiter cette exploration. Si on considère l'ensemble des solutions possibles, elle est représentable sous la forme d'un arbre. Cet arbre est vraiment énorme et on peut soit couper des branches avant d'arriver aux feuilles (optimisation classique), ou bien choisir de ne pas parcourir certaines branches (utilisation d'heuristiques).

Pour illustrer ces principes, prenons un rectangle de 3×2 avec 2 formes `Rect` et 2 `Corner` de l'autre problème. La racine de l'arbre correspond à aucune pièce placée. Les fils de la racine correspondent à toutes les possibilités de placement de ces 4 pièces. La racine a donc 24 fils. Chacun de ces fils a 18 fils correspondant à tous les placements des trois formes restantes, etc. jusqu'à placer toutes les formes. La hauteur de l'arbre est donc de 4 et le nombre de noeuds est $24 + 24 \times 18 + 24 \times 18 \times 12 + 24 \times 18 \times 12 \times 6 = 36744$.

La formule générale avec X formes et N cases est :

— nombre de noeuds : $\sum_{i=1}^{X} N^i \times \prod_{j=0}^{i-1}(X - j)$.

— nombre de feuilles : $N^X \times !X$.

Pour un « petit » rectangle de 8×5 avec 12 formes, cela donne la bagatelle d'environ 8.03×10^{27} feuilles, donc encore plus de noeuds à évaluer. Bref, c'est impossible : l'âge de l'univers est d'environ 4.4×10^{26} nanosecondes.

En reflechissant un minimum, on s'aperçoit que l'on peut couper un très grand nombre de branches prématurément. Par exemple, il ne sert à rien d'explorer une branche plus en avant à partir du moment où le noeud courant correspond à un placement impossible d'une forme, c'est-à-dire si elle chevauche une autre ou bien sort du rectangle. De plus, il est inutile d'explorer des branches qui sont des permutations dans l'ordre de placement des pièces, aboutissant toutes à une même solution. Sur un 8×5 avec 12 formes, cela conduit à évaluer « seulement » 10471978251 placements.

Malheureusement, selon l'ordre dans lequel les formes sont prises en compte, cela peut prendre environ 22 minutes pour parcourir toutes ces solutions. Il faut donc trou-

ver d'autres façons de couper les branches prématurément. Un exemple possible est de vérifier si le volume de la forme à placer ne dépasse pas le nombre de cases libres dans le rectangle. Il existe d'autres solutions, qui coupent plus de branches encore plus tôt.

Quand malgré tout, le temps d'exploration reste grand, on a recours à des heuristiques, qui vont ignorer des parties entières de l'arbre. La contrepartie, c'est que l'on ne peut pas trouver toutes les solutions optimales, voire même n'en trouver aucune, seulement une approchée.

Un exemple d'heuristique utilisable est de prendre pour hypothèse qu'il existe au moins une solution optimale comportant une forme particulière. Par exemple, dans le 8×5 avec 12 formes, on peut estimer qu'il y a de grandes chances de trouver une solution optimale avec au moins un `Rect`. On ne teste donc pas les solutions ne comportant aucun `Rect`, ce qui réduit notablement le temps. Mais si cette supposition n'est pas vraie, c'est-à-dire aucune solution optimale ne comporte de `Rect`, alors on ne trouvera pas un remplissage optimal.

Bien entendu, on peut faire des hypothèses un peu plus hasardeuses, comme le fait qu'il y a forcément des solutions optimales comportant deux `Rect`. Par exemple, sur un 8×5 avec 12 formes, il y a 2016 solutions optimales (en considérant toutes les formes comme distinctes). On peut donc faire le pari qu'il y en a forcément qui comportent deux `Rect`, et l'on aurait raison puisque toutes en comportent deux. On peut même parier sur trois `Rect` car il y en a 1728 qui en ont 3 !

Malheureusement, ce genre d'heuristique ne permet pas d'éliminer énormément de branches, et bien souvent, on est amené à faire des choix drastiques qui ne mènent pas à l'optimal.

2 l'énoncé

L'objectif est presque identique à l'autre problème : réaliser une fonction qui calcule le meilleur remplissage possible d'un rectangle, **en temps limité à 150 secondes**. Il ne faut donc plus trouver combien il existe de remplissages optimaux, ni considérer les formes identiques comme distinctes.

ATTENTION : comme pour l'autre problème, les dimensions du rectangle ainsi que le nombre et types de formes seront fournis sur l'entrée standard avec le même format, et sont donc inconnus de vous. Indication primordiale, cela sera des problèmes relativement identiques en complexité à celui du 8×5 avec 12 formes. Vous êtes donc assurés qu'une version naïve prendra plus de deux minutes et échouera.

Vous êtes donc obligé de faire de l'optimisation et/ou d'utiliser des heurisitiques. Mais attention, dans le deuxième cas, vous n'atteindrez peut être pas l'optimal. Or, deuxième indice, si vous trouvez les bonnes méthodes d'optimisation, vous pouvez écrire un code qui trouve un optimal à coup sur en moins de deux minutes, et au mieux, en quelques secondes.

Conseil : entrainez-vous avec les deux fichiers de test fournis avec le canevas. La version naïve du 7×5 s'exécute en environ 20 secondes (optimal à 32) et celle du 8×5

en environ 22 minutes (optimal à 37). Si vous réussissez à optimiser votre code pour que le 7×5 tourne en moins de 2 secondes et le 8×5 en deux minutes (selon puissance de machine), votre programme est potentiellement bon pour la soumission.

En sortie, votre programme doit afficher sur la sortie standard $N+1$ lignes de texte, N étant le nombre de formes à placer. La première ligne doit indiquer le volume occupé en nombre de cases par votre solution. Ensuite, sur chaque ligne est indiqué l'id (cf. canevas de code) d'une des formes puis ses coordonnées de placement, avec `-1 -1` si elle n'est pas utilisée. NB : Les ids n'ont pas forcément besoin d'être dans l'ordre. La table 1 donne un exemple d'entrée et la sortie associée au placement de la figure 1.

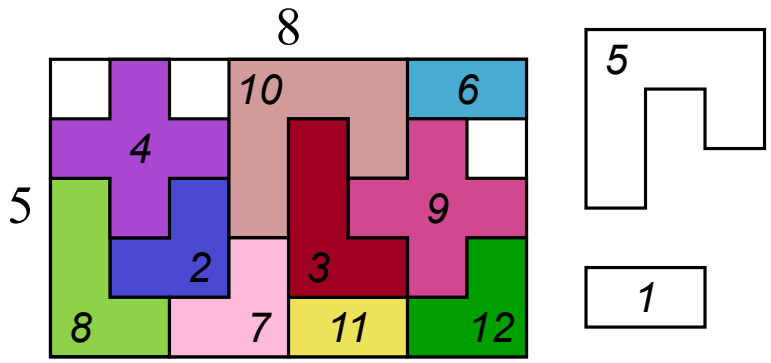

FIGURE 1 – Exemple de solution optimale avec 12 formes sur un rectangle 8×5

entrée	sortie
8 5 12 1 2 3 4 5 1 2 3 4 5 1 2	37 1 -1 -1 2 1 2 3 4 1 4 0 0 5 -1 -1 6 6 0 7 2 3 8 0 2 9 5 1 10 3 0 11 4 4 12 6 3

TABLE 1 – Exemple d'entrée avec 12 formes à placer dans un rectangle de 8×5, et une sortie possible.

D'après l'entrée, si on numérote les formes dans l'ordre de la ligne, alors la première forme est un `Rect` qui reçoit l'id 1, la deuxième est un `Corner` qui reçoit l'id 2, etc.

Pour la sortie, les coordonnées sont celles de coin supérieur gauche de la boite englobante de la forme. Pour les `Corner` et `Cross`, cela ne correspond donc pas à une case qu'elles occupent réellement. Par exemple, la forme d'id 9 est une `Cross` et ses coordonnées sont `5 1` car son coin supérieur gauche est à la colonne 5, ligne 1.

3 les ressources

Pour vous aider dans la réalisation du programme, vous trouverez sur
`http://cours-info.iut-bm.univ-fcomte.fr`
un article dans la section `hackaton` de l'année courante, portant le même titre que l'exercice. Il contient un lien permettant de télécharger un canevas de code, ainsi que le fichier d'entrée donné ci-dessus.

Bien entendu, vous êtes libres d'utiliser ou non ce canevas, mais c'est un gain de temps que de s'en servir comme base.