Projet de programmation
Rapport

ENSAE - 2024

Martin DUFFAUD & Donatien Roi

L'ensemble du code du projet est disponible sur le dépôt
https://github.com/martin-duffaud/ensae-prog24

Table des matières

1 Séances 1 et 2

1.1 Question 3

1.1.1 Complexité et terminaison

research est en $O(mn)$, relocate fait appel a research puis réalise au plus $m+n$ swaps, donc est en $O(mn+m+n) = O(mn)$. Comme *get_ solution* fait appel mn fois a relocate, elle est en $O((mn)^2)$. Comme research, qui est appelée mn fois réalise au plus m+n swaps, la longueur de la solution est $mn * (m+n)$.

1.1.2 Optimalité et terminaison

Comme l'algorithme replace successivement $1, ..., mn$ dans cet ordre, la fonction termine et donne une solution valide. Néanmoins, comme les opérations sur chacun des éléments $1, ..., mn$ sont indépendantes, dans le sens ou le déplacement de i, lorsque $1, ..., i-1$ sont déjà rangés, peut modifier les positions des éléments non encore rangés, elles ne cherchent pas nécessairement à faire les déplacements les plus adéquats, tant que l'élément i arrive au bon

endroit. Ainsi, un nombre important d'opérations ne sont pas exploités, voire inutiles. On pourrait, pour améliorer l'algorithme, essayer de décider, en fonction des swaps possibles pour permettre a i d'arriver au bon endroit, de choisir le déplacement qui aurait l'influence la plus positive sur les éléments non encore rangés.

1.1.3 Remarques supplémentaires

Une autre proposition d'algorithme, considérée comme "naïve" par le binôme est disponible également sous le nom de *get_ solution2* dans la classe Solver. Cette propostion avait déjà été évoqué avec le chargé de TD, et donc ne fera pas l'objet de précisions supplémentaires, sauf demande du correcteur après le premier rendu. Notez seulement que *get_ solution* donne une solutions avec une meilleure complexité, en $O(mn * (max(m + n)))$ selon nos calculs.

1.2 Question 6

Les noeuds étant des grilles qui sont une réorganisation de l'ensemble $1, ..., mn$, on remarque naturellement la proximité d'un tel outil avec l'ensemble des permuations sur $1, ..., mn$, noté S_{mn}. Ainsi, on a choisit d'implémenter les fonctions suivantes pour la suite :

- *permutation* , qui est une fonction récursive, qui étant donné une grille de taille $m * n$, renvoie l'ensemble des éléments de S_{mn}.
- *gridlist_ from_ permlist* qui permet de construire la liste des grilles à partir de la liste des permutations.
- *all_ swaps_ possible* qui renvoie étant donné une grille, l'ensemble des swaps possible depuis cette grille.

1.3 Question 7

Techniquement, *construct_ graphe* a une complexité en $O((n * m)!^3)$ car on fait un while sur toutes les grilles possibles de taille $n * m$ (en temps (n*m) !), puis on les parcourt toutes (temps en $(n * m)!$), puis on parcourt tous les noeuds du graphe (temps encore en (n*m) !). Le graphe construit a $(n * m)!$ noeuds (qui correspondent aux grilles et donc aux permutations de l'ensemble $1, 2, ..., n * m$) et comme c'est un arbre il a $(n * m) - 1$ arêtes.

2 Séances 3 et 4

2.1 Question 2

Un autre exemple d'heuristique possible pour astar est donnée par la distance de *Kendall-Tau* Comme les grilles sont des permutations sur $1, ..., mn$, la distance de Kendall-Tau, d_{KT}, qui donne le nombre de paires d'éléments qui sont en désaccord par rapport à leur ordre. En particulier, dans notre cas, on compare une grille avec l'identité, donc on va simplement calculer le nombre d'inversion d'une permutation. Si g est une grille et Id est l'identité, le nombre d'inversion de g est $d_{KT}(g, Id)$.

$$\forall g \in S_{mn}, d_{KT}(g, Id) = Card((i, j) \in 1, ..., mn : i < j \wedge g(i) > g(j))$$

En pratique, la distance de Kendall Tau est équivalente au nombre de swaps effectués par le Tri à Bulle. Elle prend ainsi ses valeurs entre 0 et mn.

L'intérêt de cette heuristique est qu'elle est par nature fondée sur le problème de recherche d'un nombre de swaps optimal entre deux grilles, puisqu'elle mesure justement cette valeur. L'implémentation naive est en $O((mn) * *2)$. Cependant, comme on compare toujours une grille avec l'identité, l'utilisation du tri fusion permet, en comptant le nombre de "sauts" des éléments de la 2e liste lors de la fusion, d'obtenir un temps de calcul en $O((mnlog(mn)))$.

Remarque : ces éléments de réponses sont rappelés en commentaire au moment d'implémenter la fonction naïve qui calcule la distance de Kendall-Tau, *kendall_tau_naif*

2.2 Question 4

Pour la question 4 et la création de niveaux de difficulté, le fait d'utiliser la distance de Kendall-Tau comme heuristique simplifie grandement le choix de grille à effectuer : on aura qu'à, à partir de l'identité, réaliser un certain nombre d'inversions pour choisir la difficulté. On remplace donc l'heuristique du code de astar (distance de Manhattan) par celle de Kendall-Tau. On suppose donc que ce changement a été effectué (ce qui revient à remplacer les lignes 336 à 343 du code par les lignes 395 à 401).
On choisit 4 niveaux de difficultés. Soit g une grille. En notant $d_{max} = mn$ (qui est la valeur maximale pour $d_{KT}(g, Id)$), on a :

- La difficulté du jeu est $EASY := 1$ ssi $1 <= d_{KT}(g, Id) <= d_{max}/4$
- La difficulté du jeu est $INTERMEDIATE := 2$ ssi $d_{max}/4 < d_{KT}(g, Id) <= d_{max}/2$
- La difficulté du jeu est $DIFFICULT := 3$ ssi $d_{max}/2 < d_{KT}(g, Id) <= 3d_{max}/4$
- La difficulté du jeu est $HARDCORE := 4$ ssi $3d_{max}/4 < d_{KT}(g, Id) <= d_{max}$

2.3 Question 5

Pour étendre le jeu, on peut par exemple en interdire certains échanges de cases, ce qui implique de directement changer la fonction *swap* implémentée dans grid. On pourrait par exemple la réécrire de la manière suivante :

```
def swap_with_barriers(self, cell1, cell2, barriers_list):

        """
        Implements the swap operation between two cells.
        Raises an exception if the swap is not allowed.

        Parameters:
        ----------
        cell1, cell2: tuple[int]
            The two cells to swap. They must be in the format (i, j)
            where i is the line and j the column number of the cell.

        barriers_list : list of tuples containing two cells in the
        format (i,j)
        """

        i1, j1 = cell1
        i2, j2 = cell2
```

```
if (i1 == i2 and abs(j1-j2) == 1 or j1 == j2 and abs(i1-i2))
and (cell1,cell2) not in barriers_list
and (cell2,cell1) not in barriers_list:

    self.state[i1][j1], self.state[i2][j2] =
    self.state[i2][j2], self.state[i1][j1]

else:
    raise Exception("Sorry, this swap is not allowed")
```

2.4 Question 6

Si la grille est $1 * n$, il s'agit juste d'un tri. Sa compléxité est en $nlog(n)$, si on prend par exemple un tri fusion. Si la grille est $k * n$, une stratégie peut être : dans un premier temps, réussir à mettre dans chaque ligne i (de 0 à $k-1$) les éléments $(i * n) + 1$ jusqu'à $(i * n) + n$, puis trier chaque ligne comme précédemment.