Agrégation externe de mathématiques, session 2008
Épreuve de modélisation, option informatique

(*public 2008*)

Résumé : Ce texte décrit et étudie le jeu de taquin, et s'intéresse en particulier à la recherche de solutions optimales.

Mots clefs : parcours de graphes

► *Il est rappelé que le jury n'exige pas une compréhension exhaustive du texte. Vous êtes laissé(e) libre d'organiser votre discussion comme vous l'entendez. Des suggestions de développement, largement indépendantes les unes des autres, vous sont proposées en fin de texte. Vous n'êtes pas tenu(e) de les suivre. Il vous est conseillé de mettre en lumière vos connaissances à partir du fil conducteur constitué par le texte. Le jury appréciera que la discussion soit accompagnée d'exemples traités sur ordinateur.*

1. Exposé du problème

1.1. *Présentation du jeu de taquin*

Le *jeu de taquin* consiste à manipuler une *table* carrée T à $n \times n$ cases, avec $n \geq 2$, sur laquelle on pose n^2-1 pièces numérotées de 1 à n^2-1. Il reste donc une seule case vide appelée *trou*. Le seul mouvement ou *coup* autorisé consiste à faire glisser l'une des pièces adjacentes au trou vers celui-ci, ce qui revient à échanger leurs positions respectives. On peut donc distinguer quatre types de coups : N (pour Nord), E (Est), S (Sud) et O (Ouest), suivant la direction dans laquelle le trou est déplacé lors d'un coup. Une table obtenue à partir d'une table T en un seul coup de type $a \in \{N,E,S,O\}$ est appelée *successeur* de T, et notée $s_a(T)$. Le but du jeu est, à partir d'une table initiale aléatoire, de reconstituer la *table finale*, notée T_f, dans laquelle les pièces sont rangées dans l'ordre naturel.

La figure 1 présente trois tables du jeu pour $n = 4$; la première, notée T_1, est aléatoire, la troisième est la table finale T_f et la deuxième, notée T_2, peut être qualifiée d'intermédiaire : il est possible de passer en un certain nombre de coups de T_1 à T_2, puis de T_2 à T_f.

10	6	4	12
1	14	3	7
5	15	11	13
8	0	2	9

table T_1

0	1	2	4
3	6	10	12
5	7	14	11
8	9	15	13

table T_2

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

table finale T_f

FIG. 1. Différentes tables du jeu de taquin pour $n = 4$.

1.2. *Représentation des tables, des coups et du jeu*

Une table T du jeu de taquin est codée par un tableau carré (encore noté T) où, pour $i, j \in [[0, n-1]]$, $T[i, j]$ désigne le numéro de la pièce située en ligne i et colonne j, le trou étant codé par l'entier 0. Par exemple, dans la table T_1 de la figure 1, $T_1[2,0] = 5$ et le trou est situé à la position $[3,1]$.

À une table T sont associées deux permutations. La première est dite *standard* et notée σ : elle porte sur l'intervalle d'entiers $[[0, n^2-1]]$ et correspond à la lecture de la table ligne après ligne de manière « normale » (de gauche à droite). La seconde est dite *par boustrophédon* et notée σ^B : elle correspond à la lecture de la table ligne à ligne mais par allers et retours successifs (la première ligne est parcourue de gauche à droite, la deuxième de droite à gauche, la troisième de gauche à droite, etc.) ; de plus, on ne tient pas compte du trou, si bien que σ^B est une permutation sur l'ensemble $[[1, n^2-1]]$.

La figure 2 représente les deux permutations σ_1 et σ_1^B associées à la table T_1 de la figure 1 et les permutations σ_1' et $\sigma_1'^B$ associées à la table $T_1' = s_N(T_1)$.

$$T_1 \qquad \begin{cases} \sigma_1 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 10 & 6 & 4 & 12 & 1 & 14 & 3 & 7 & 5 & 15 & 11 & 13 & 8 & 0 & 2 & 9 \end{pmatrix} \\ \sigma_1^B = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 10 & 6 & 4 & 12 & 7 & 3 & 14 & 1 & 5 & 15 & 11 & 13 & 9 & 2 & 8 \end{pmatrix} \end{cases}$$

$$T_1' = s_N(T_1) \qquad \begin{cases} \sigma_1' = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 10 & 6 & 4 & 12 & 1 & 14 & 3 & 7 & 5 & 0 & 11 & 13 & 8 & 15 & 2 & 9 \end{pmatrix} \\ \sigma_1'^B = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 10 & 6 & 4 & 12 & 7 & 3 & 14 & 1 & 5 & 11 & 13 & 9 & 2 & 15 & 8 \end{pmatrix} \end{cases}$$

FIG. 2. Permutations associées aux tables T_1 et $s_N(T_1)$.

Le jeu de taquin peut être modélisé par un graphe orienté étiqueté G : l'ensemble des sommets est l'ensemble des $(n^2)!$ tables associées aux permutations de l'ensemble $[[0, n^2-1]]$; un arc d'étiquette $a \in \Sigma = \{N, E, S, O\}$ relie deux tables T et T' si et seulement si $T' = s_a(T)$. L'étiquette d'un chemin entre deux sommets du graphe G est alors le mot sur l'alphabet Σ obtenu par la concaténation des étiquettes de chacun des arcs de ce chemin en suivant leur ordre de parcours.

Une table T est dite *jouable* s'il existe un chemin dans G entre T et la table finale T_f. Une *solution optimale* du jeu de taquin à partir d'une table T jouable est un mot sur l'alphabet Σ étiquetant un chemin de longueur minimale entre T et T_f.

Il existe des tables non jouables : par exemple, pour $n = 4$, la table obtenue à partir de la table finale T_f en permutant les pièces de numéros 1 et 2 ne permet pas, avec les règles du jeu, de reconstruire T_f. Le graphe G possède deux composantes connexes et seules les tables appartenant à la composante qui contient T_f sont jouables. On dispose même du critère de décision suivant : une table T est jouable si et seulement si la permutation par boustrophédon σ^B associée à T est paire. Tester si une table est jouable est alors très simple.

2. Premières stratégies de résolution

Il s'agit maintenant de concevoir et d'étudier différentes stratégies de résolution du problème du taquin (dans le cas d'une table jouable). La recherche systématique d'une solution optimale est coûteuse et d'autres méthodes permettent d'obtenir plus rapidement un chemin dans G qui, sans être de longueur excessive, n'est pas forcément de longueur minimale.

On présente ici deux stratégies de résolution du jeu de taquin, relativement simples mais tout à fait différentes.

2.1. *Méthode ligne à ligne*

Une stratégie assez rudimentaire consiste à placer correctement les $(n-2)$ dernières lignes de la table successivement de la dernière à la troisième et, pour les deux premières lignes restantes, de placer correctement les n colonnes, de la dernière à la première. On peut s'assurer qu'un tel algorithme est correct. Sa grande simplicité et son coût peu onéreux en sont les points forts ; il correspond d'ailleurs à la stratégie vulgarisée dans le grand public car, ne nécessitant presque aucune mémoire, il peut aisément être exécuté par un humain. Mais il fournit rarement des solutions optimales.

2.2. *Exploration du graphe*

Un autre algorithme basé sur une idée très simple consiste à explorer le graphe G à partir de la table initiale T. Cette exploration peut alors se faire selon les deux stratégies suivantes :

2.2.1. *Exploration en largeur.* La stratégie la plus naturelle consiste à explorer le graphe G en largeur à partir de la table initiale T. Il faut alors marquer les tables déjà visitées afin d'éviter les calculs redondants et l'algorithme s'arrête dès que l'on rencontre la table finale T_f. Cela nécessite en particulier de gérer une file d'attente des tables dont il faudra calculer les successeurs. L'avantage de cette méthode, outre sa simplicité de conception et de programmation, est qu'elle fournit systématiquement une solution optimale. Son inconvénient en est le coût, au niveau de la complexité en temps mais aussi en espace.

2.2.2. *Exploration en profondeur à seuils progressifs.* Est-il possible d'améliorer cette complexité par une exploration en profondeur du graphe G ? De prime abord, une telle méthode ne semble pas beaucoup plus satisfaisante : si la file d'attente dans l'exploration en largeur est ici remplacée par la pile des tables situées en amont sur le chemin d'exploration (de taille beaucoup plus modeste a priori), il n'en reste pas moins vrai qu'il faut toujours marquer les tables déjà visitées pour éviter l'exploration de branches infinies dues à la présence de cycles. Un stratagème pour éviter ce marquage coûteux sans mettre en péril la terminaison de l'algorithme consiste à stopper l'exploration à une certaine profondeur k. Mais comme on ne sait pas, en général, majorer de manière efficace la longueur d'un chemin optimal, il faut envisager d'effectuer plusieurs parcours en profondeur successifs, pour des valeurs croissantes du seuil k. Cette exploration peut paraître coûteuse, car on parcourt de nombreuses fois les tables proches de la table initiale

T ; mais l'essentiel du calcul a lieu loin de T et il est établi que ce défaut n'a pas de conséquence néfaste sur la complexité globale de l'algorithme.

3. Améliorations de l'exploration

Dans les stratégies d'exploration du graphe G mises en place au § 2.2, la recherche de la table finale T_f se fait en force brute – à l'aveugle – comme si tous les coups se valaient, ce qui ne correspond manifestement pas à la réalité.

3.1. *Objectifs poursuivis*

On souhaite donc à présent améliorer les algorithmes des paragraphes 2.2.1 et 2.2.2 avec les deux objectifs suivants :

1) développer des techniques permettant d'ordonner la liste des successeurs d'une table donnée par des critères d'efficacité de manière à pouvoir orienter la recherche dans le graphe G dans la direction de la table T_f ;
2) faire en sorte qu'une stratégie d'exploration du graphe G reposant sur l'amélioration obtenue au point 1) permette d'élaguer (dans la plupart des cas) l'arbre d'exploration.

3.1.1. *Fonction de poids sur le graphe.* Pour atteindre le premier objectif, imaginons disposer d'une *fonction de poids* π définie sur l'ensemble des sommets du graphe G et à valeurs dans $\mathbb{N}$, qui mesurerait, pour toute table T, une certaine distance $\pi(T)$ de T à la table T_f (évidemment, $\pi(T_f) = 0$). À partir d'une table T quelconque, on préférerait alors franchir en priorité un arc menant à une table T' telle que $\pi(T') < \pi(T)$, car un tel arc nous rapproche de T_f. Ceci revient en fait à étiqueter les arcs reliant deux sommets T et T' du graphe non plus seulement par la lettre $a \in \Sigma$ telle que $T' = s_a(T)$, mais par le couple lettre-poids $(a, \pi(T') - \pi(T)) \in \Sigma \times \mathbb{Z}$; dans une stratégie glouton, au coup par coup, un meilleur arc issu de T est alors celui de poids le plus petit (en valeur algébrique). Le poids d'un chemin dans le graphe est défini comme la somme des poids des arcs qui le constituent. Notons qu'alors tout cycle dans le graphe est de poids nul.

On propose ici deux fonctions de poids π_1 et π_2 sur l'ensemble des somments (on pourrait en imaginer d'autres). Pour une table T de G :

- $\pi_1(T)$ est égal au nombre d'inversions de la permutation standard σ associée à la table T (c'est-à-dire le nombre de couples (i, j) de $[[0, n^2-1]]^2$ tels que $i < j$ et $\sigma(i) > \sigma(j)$) ;
- $\pi_2(T) = \sum_{0 \leq i,j \leq n-1, T[i,j] \neq 0} d(T[i,j])$, où $d(T[i,j])$ est la distance pour la norme 1 entre l'emplacement de la pièce de valeur $T(i,j)$ dans la table T et l'emplacement de cette même pièce dans la table finale T_f ; le trou n'est pas concerné par cette sommation. Dans la table T_1 de la figure 1, par exemple, $d(T_1[0,3]) = 6$, car la pièce numéro 12 doit être déplacée de 3 cases vers le bas et de 3 cases vers la gauche pour atteindre le plus rapidement possible sa position dans la table finale T_f ; $\pi_2(T_1) = 38$.

Exercice de programmation :

Il vous est demandé de rédiger un programme conforme aux spécifications ci-dessous dans l'un des langages C, Caml ou Java à votre choix. Ce programme devra être accompagné d'un exemple d'éxécution permettant d'en vérifier le bon fonctionnement. La clarté et la concision du programme seront des éléments importants d'appréciation pour le jury.

Implanter dans l'un des langages autorisés l'algorithme qui, à partir d'une table T du jeu de taquin, calcule les coordonnées du trou et la valeur de $\pi_2(T)$.

3.1.2. *Fonction de distance minorante sur le graphe.* L'atteinte du second objectif repose sur une propriété intrinsèque au jeu de taquin, qui consiste à savoir minorer (de manière non triviale, bien sûr) la longueur d'un chemin optimal d'une table T à la table T_f. Pour simplifier, on expose l'intérêt d'une telle propriété sur un exemple plus concret que le jeu de taquin.

Supposons disposer d'un graphe dont les sommets sont certaines villes de France et tel qu'un arc entre deux sommets a pour étiquette la (meilleure) distance routière entre les deux villes qu'il relie. On cherche par exemple une route optimale entre Paris et Brest. Comme la distance routière entre ces deux villes peut raisonnablement être considérée comme inférieure à la somme de la distance routière entre Paris et Lyon et de la distance à vol d'oiseau entre Lyon et Brest, il y a de grandes chances qu'aucun chemin passant par Lyon ne doive être exploré (notons cependant que ceci ne serait pas le cas si, par exemple, la France ne possédait que deux routes : une de Paris à Lyon et une de Lyon à Brest). Le bien fondé de cette remarque repose évidemment sur le fait qu'une distance routière entre deux villes est toujours minorée par la distance à vol d'oiseau entre ces villes. Ce raisonnement nécessite donc la connaissance d'une *fonction de distance minorante* de la distance routière : la distance à vol d'oiseau en l'occurrence.

Cette idée peut être adaptée au jeu de taquin, si l'on dispose d'une fonction de distance minorante intéressante sur le graphe G : celle-ci doit minorer (mais pas trop, évidemment !) le nombre optimal de coups permettant de passer d'une table T à la table T_f. Il se trouve qu'une telle fonction existe.

3.2. *Explorations plus performantes du graphe*

Il s'avère que les deux outils présentés au § 3.1 ont un impact très différent sur les deux algorithmes d'exploration du graphe décrits au § 2.2 : alors qu'ils sont d'un intérêt très limité pour l'exploration en largeur, ils peuvent en revanche améliorer nettement l'exploration en profondeur par seuils progressifs.

3.3. *Possibilité d'un algorithme glouton ?*

Le bon comportement de l'algorithme en profondeur avec une bonne fonction de poids π pourrait même laisser espérer un moment que l'algorithme glouton, consistant à choisir à chaque

étape le meilleur coup possible vis-à-vis de π, aboutisse directement à la table T_f : un seul chemin dans le graphe serait alors parcouru et fournirait une solution optimale (pour π). Malheureusement, il n'en est rien en général, comme on peut le constater par l'examen de la table T_2 avec la fonction de poids π_1.

On pourrait cependant imaginer des mécanismes pour échapper à un minimum local (vis-à-vis de la fonction de poids π) lorsque cet algorithme glouton échoue.

Suggestions pour le développement

► *Soulignons qu'il s'agit d'un menu à la carte et que vous pouvez choisir d'étudier certains points, pas tous, pas nécessairement dans l'ordre, et de façon plus ou moins fouillée. Vous pouvez aussi vous poser d'autres questions que celles indiquées plus bas. Il est très vivement souhaité que vos investigations comportent une partie traitée sur ordinateur et, si possible, des représentations graphiques de vos résultats.*

- Exprimer le nombre d'arcs du graphe G en fonction de l'entier $n \geq 2$.
- Prouver le critère énoncé par le texte à la fin du § 1.2 pour tester si une table T est jouable ou pas. En déduire le nombre de sommets et d'arcs utiles dans le graphe G.
- Certaines séquences de coups (par exemple *NS* ou *EO*) amènent systématiquement à explorer un cycle dans le graphe G. Deux séquences de coups peuvent également être *équivalentes*, c'est-à-dire conduire systématiquement à la même table dans G ; exhiber par exemple une séquence équivalente à *ENOSEN*. Proposer des mécanismes pour adapter les algorithmes du texte de manière à éviter certaines séquences de coups appartenant à une liste établie à l'avance.
- Concernant les algorithmes d'exploration du graphe :
  - Estimer la complexité en temps et en espace de l'algorithme en largeur (§ 2.2.1) ; proposer une amélioration consistant à explorer le graphe G à la fois à partir de la table initiale T et de la table finale T_f et estimer le gain alors réalisé.
  - Justifier que la part des calculs répétés plusieurs fois par l'algorithme d'exploration en profondeur à seuils progressifs (§ 2.2.2) n'en altère pas la complexité globale et développer une stratégie d'obtention d'une solution optimale.
  - Faire une étude comparée de ces deux algorithmes avec un (ou deux) algorithme(s) connu(s) de recherche de plus court chemin dans un graphe.
- Concernant les fonctions de poids $\pi = \pi_1$ ou π_2 (§ 3.1.1), proposer une méthode de calcul incrémental, consistant à déterminer $\pi(s_a(T))$ en fonction de $\pi(T)$ et non pas directement ; estimer l'apport d'une telle modification en termes de complexité.
- Proposer une fonction de distance minorante intéressante (§ 3.1.2).

- Concernant l'algorithme glouton (§ 3.3) :
  - Montrer qu'il est correct pour $n = 2$, en représentant la partie utile du graphe G ; préciser dans ce cas le nombre de coups maximal d'une solution optimale.
  - Indiquer quel(s) mécanisme(s) on peut imaginer pour échapper à un minimum local lorsque l'algorithme glouton échoue.
- Exhiber, pour $n = 4$ par exemple, une table T pour laquelle l'algorithme ligne à ligne (§ 2.1) fournit une solution qui est manifestement loin d'être optimale.