MATH0499-1 Théorie des Graphes

Enumération de circuits et chemins hamiltoniens
Le parcours du cavalier

Mazurchyk Aliaksei
Antoine Sadzot

15 décembre 2017

Université de Liège

Table des matières

1 Présentation du projet

1.1 Énoncé

Lorsque nous avons sélectionné notre projet, l'énoncé d'origine était le suivant :
« Étant donné un échiquier de dimension $m \times n$, énumérer tous les circuits hamiltoniens réalisés en utilisant uniquement les mouvements du cavalier dans le jeu d'échecs. Même question, mais en énumérant le nombre de chemins hamiltoniens "ouverts" (origine et destination différentes). »

A cet énoncé, nous avons décidé d'ajouter une autre partie :
« Étant donné un échiquier de dimension $m \times n$, générer et afficher un circuit (ou un chemin) hamiltonien basé uniquement sur les mouvements du cavalier. »

1.2 Les bases du problème du cavalier

Le problème du cavalier peut donc être représenté sous la forme d'un graphe. Chaque case de l'échiquier est un sommet et chaque mouvement possible à partir de cette case vers une autre case est une arête du graphe.

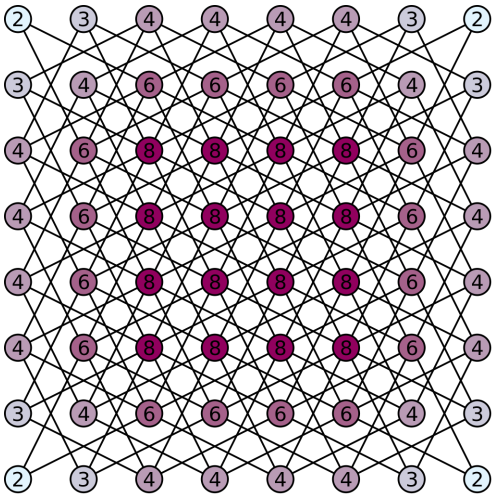

FIGURE 1 – Graphe du cavalier sur un échiquier 8×8(inclure source ici wiki)

Le graphe de la figure 1 représente les déplacements possibles pour un cavalier sur un échiquier classique. Le chiffre attribué à chaque case représente le nombre de cases accessibles en un mouvement depuis celle-ci. La réalisation de ce travail implique la génération d'un graphe similaire, généralisé à n'importe quelle taille $m \times n$.

Le graphe ainsi généré peut ensuite être exploité pour y trouver des chemins et circuits hamiltoniens.

2 Recherches

2.1 Condition d'existence

Allen J. Schwenk démontra en 1991 que trois conditions empêchent la réalisation d'un tour de cavalier sur un échiquier rectangulaire $m \times n$ pour tout $m \leq n$. Si aucune de ces condition n'est remplie alors au moins un circuit hamiltonien existe (par extension un chemin hamiltonien existe également).

1. m et n sont impaires
2. $m = 1, 2$, ou 4
3. $m = 3$ et $n = 4, 6$ ou 8

2.2 Algorithmes de parcourt

Le problème du cavalier étant connu et étudié depuis plusieurs siècles, un certain nombre d'algorithmes existent pour trouver un circuit ou un chemin hamiltonien.

2.2.1 Force brute

Cette méthode permet de trouver un chemin/circuit à coup sûr si un tel chemin existe. Le principal inconvénient de cette technique c'est qu'elle nécessite un long temps de calcul. Pour trouver un chemin/circuit, nous établissons un arbre de parcours. La racine de l'arbre est la case de départ. Les branches représentent les déplacements valides possibles vers les autres cases. L'algorithme doit parcourir tout l'arbre des possibilités. S'il tombe sur un cul de sac (c'est à dire si le sous-arbre parcouru est de hauteur inférieure au nombre de cases de l'échiquier), il revient en arrière. Tous les chemins possibles sont ainsi testés.

Trouver un seul chemin/circuit peut prendre quelque secondes ou minutes si l'échiquier est suffisemment petit. A ce moment là, l'algorithme de parcourt s'arrête et la solution est affichée. L'énoncé cependant implique de trouver tous les chemins ou circuits sur l'échiquier afin de les énumérer. Ceci augmente grandement les temps de calcul. Pour un échiquier de taille 8×8, il y a 4×10^{51} suites de mouvements possibles. Ce qui conduit à des temps de calculs trop importants pour les machines actuelles. Cette méthode est la seul connue pour énumérer tous les chemins et circuits.

L'encyclopédie en ligne des suites de nombres entiers (OEIS) comporte la liste du nombre de chemins hamiltoniens en fonction de la taille d'un échiquier carré. Cette suite nommée A165134 permet de comparer les résultats obtenus expérimentalement avec les résultats officiels afin de valider l'efficacité de l'algorithme.

2.2.2 Warnsdorf

La loi de Warnsdorff est une heuristique développée par H. C. von Warnsdorff en 1823 qui permet de trouver un chemin hamiltonien sur un échiquier carré. L'heuristique se base sur le graphe présenté en figure 1 où chaque nœud se voit attribuer un poids correspondant au nombre de nœuds accessibles. On démarre le chemin sur un nœud au hasard. De ce nœud, on se déplace vers le nœud de poids le plus faible. Si plusieurs nœuds de même poids sont accessibles, ceux-ci doivent être départagés. La méthode a l'avantage d'avoir une complexité linéaire en aire (n*n).

La méthode de départage la plus simple est la sélection aléatoire du nœud suivant. Cette méthode peut paraître simpliste, mais elle est pourtant efficace jusqu'à une certaine taille d'échiquier.

De nombreux mathématiciens se sont basés sur la loi de Warnsdorff pour y incorporer leur propre méthode de départage. Nous nous sommes plus particulièrement intéressés aux travaux de D. Squirrel, qui a élaboré en 1996 une méthode pour départager les candidats. Celle-ci permet de trouver à coup sûr un chemin hamiltonien pour toute taille d'échiquier carré, si tant est que celui-ci existe. Pour ce faire, toutes les cases accessibles depuis une case sont numérotées de 1 à 8 dans le sens horlogique, la case 1 étant celle en haut à gauche et la 8 en haut à droite. Selon la taille de l'échiquier et la position du cavalier, un ordre de parcours est définit. Par exemple : 34261578. Le chiffre à gauche est la première case à visiter et celle à droite est la dernière. Lorsque plusieurs cases ont le même poids, elles sont départagées par cet ordre de parcours. Nous avons retranscrit l'algorithme proposé par Squirrel dans un programme en C. Nous avons pu observer la fiabilité de ses résultats. Sa méthode a également une complexité O(n*n).

2.2.3 Echiquier rectangulaire

La forme de l'échiquier influence évidemment les méthodes de recherches du problème du cavalier. Il est à noter que la loi de Warnsdorff avec départage aléatoire peut fonctionner sur des échiquiers rectangulaires, tel que nous l'avons testé. Néanmoins, l'heuristique présente évidemment les mêmes faiblesses que pour les échiquiers carrés.

Nous somme attardés sur les recherches de Shun-Shii Lin et Chung-Liang Wei, qui ont réussi à créer un algorithme pour trouver à coup sur un chemin ou un circuit Hamiltonien sur un échiquier rectangulaire de taille m*n. A condition qu'un tel chemin existe. Ils se sont basés sur les recherches de Parberry, qui a développé un algorithme basé sur le principe du "diviser pour régner". Avec ce principe, l'échiquier est divisé en petits échiquiers. Ensuite, un chemin hamiltonien est trouvé pour chaque petit échiquier. Ensuite, les chemins sont reliés ensemble pour former un seul grand échiquier. Par manque de temps, nous n'avons implémenté qu'une partie de l'algorithme, permettant de résoudre le problème avec un échiquier de taille 3*m. Encore une fois avec une complexité O(n*m).

3 Implémentation

3.1 BruteForce

L'implémentation de l'algorithme d'énumération de chemins et circuits par force brute utilise la librairie de gestions de graphes mise à disposition des étudiants sur le site discmath. Toute interaction de l'application se fait en ligne de commande via les arguments de l'exécutable. L'utilisateur peut ainsi choisir le type de parcourt ouvert ou fermé ainsi que la taille de l'échiquier en X et en Y. Cela permet la création de scripts pour automatiser l'exécution qui peut prendre un temps considérable pour des tailles importantes d'échiquier. Lors de l'initialisation, un graphe représentant les cases accessibles à partir de chaque case est crée. Ensuite une fonction récursive parcourt ce graphe en fonction de la liste d'adjacence de chaque nœud et des cases déjà visitées.

Cet algorithme de parcourt récursif est appelé pour chaque case de départ possible. Cela permet de créer un tableau montrant le nombre de chemins existants en fonction de la position de départ. L'utilisateur peut voir l'évolution du calcul en temps réel dans le terminal.

TABLE 1 – Tableau crée pour une recherche de chemins sur un échiquier de taille 5*5

304	0	56	0	304
0	56	0	56	0
56	0	64	0	56
0	56	0	56	0
304	0	56	0	304

Afin de minimiser les temps de calcul, le lancement de la fonction de parcourt se fait en multithreading via l'API OpenMP. Celle-ci est intégrée au compilateur gcc. Les threads générés se partagent les cases de départ. De cette maière, les temps de calcul peuvent-être diminués.

TABLE 2 – Temps de calcul pour des grilles carrées

Taille de l'échiquier	4	5	6
Temps de calcul	0,005 s	1,5 s	560m 30s

3.2 Génération de chemins et circuits hamiltoniens

Pour la partie du code concernant la génération de circuits et chemins sur des échiquiers de taille variable, nous avons implémenté trois fonctions, pour les trois algorithmes sélectionnés :

- Heuristique de Warnsdorff pour trouver un chemin ou un circuit sur un échiquier carré ou rectangulaire avec départage aléatoire.
- Algorithme de Squirrel pour trouver à coup sur un chemin sur un échiquier carré.
- Algorithme de Shun-Shii pour le cas d'un échiquier de trois lignes et m colonnes. Permettant de trouver un chemin hamiltonien.

3.2.1 Génération de parcours et chemins

Dans ces fonctions, nous enregistrons l'ordre de parcours des cases dans un vecteur A. Le premier élément de ce vecteur représente la case en haut à gauche. Le dernier élément représente la case en bas à droite. Comme si les rangées de l'échiquier avaient été mises bout à bout. Les éléments contiennent comme valeur l'ordre de passage du cavalier sur la case correspondante. Allant donc de 0 à n*m-1. Un vecteur B contient le poids des cases, comme sur la figure 1. Il est à noter que tous nos algorithmes commencent dans la case en haut à droite de l'échiquier. Théoriquement, l'algorithme avec départage aléatoire pourrait commencer sur n'importe quelle case. Mais par facilité, nous avons décidé arbitrairement de celle-ci.

Au début, nous avions décidé d'utiliser l'interface de génération de graphes vue au cours, en la modifiant légèrement. Cependant, le résultat obtenu avec l'algorithme de Warnsdorff était trop compliqué et trop lent. Nous avons donc décidé de nous inspirer de l'algorithme de Squirrel, basé sur les matrices décrite au paragraphe précédent. Le fonctionnement des algorithme a été décrit au chapitre précédent.

3.2.2 Test de la validité des parcours et chemins

Pour vérifier la justesse des parcours et chemins générés, nous avons écrit deux fonctions de test. L'une pour les parcours et l'autre pour les chemins. Leur fonctionnement est pratiquement

identique. L'algorithme commence à la case "0". On regarde ensuite si la case "1" est accessible en un mouvement de cavalier. Si oui, on s'y déplace,... Si toutes les cases ont pu être visitées, le chemin est valide. Pour le test de circuits, on vérifie en plus si la case "0" est accessible depuis la case "n*n-1".

3.2.3 Affichage du résultat

En plus de la génération, nous proposons d'enregistrer le résultat obtenu dans un fichier texte. La figure 2 montre plusieurs exemples de parcours générés.

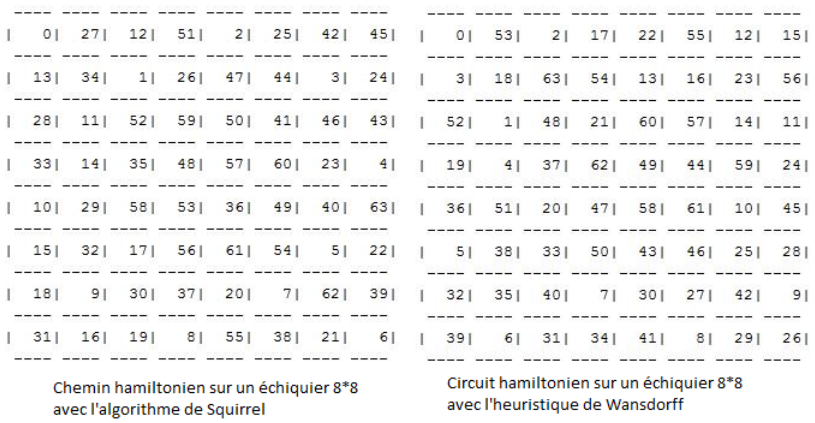

FIGURE 2 – Exemple de chemins et circuits générés

4 Tests de performance

4.1 Génération de chemins et circuits hamiltoniens

Dans ce chapitre, nous allons parler des résultats obtenus avec les algorithmes de génération ce chemins et circuits.

4.1.1 Chemins et circuits avec l'heuristique de Warnsdorff

Pour trouver des chemins, le départage aléatoire de candidats de même poids sous l'heurisitque de Warnsdorff fonctionne bien avec de "petits échiquiers carrés". En dessous de 50 cases de côté, l'heuristique permet de trouver un chamin à presque tous les coups. Et ce, dans des temps inférieurs à la seconde. Cependant, pour de plus grands échiquiers, l'heuristique est bien moins fiable. Le caractère aléatoire du départage necessite parfois plusieurs générations pour trouver un chemin valide. La figure 3 représente l'évolution de l'efficacité de l'heuristique. Pour chaque taille de tableau, 100 générations de chemins ont été effectuées. La proportion de chemins valides est représentée sur le graphique. ON observe effectivement qu'au delà de 50 cases de côté, l'efficacité dégringole.

Des tests ont également été faits avec des échiquiers rectangulaires, de taille n*m, n<=m. La figure 4 présente un graphique du taux de réussite. Chaque échiquier testé est représenté avec son nombre de lignes en abscisse et son nombre de colonnes en ordonnée. Si la couleur de l'échiquier

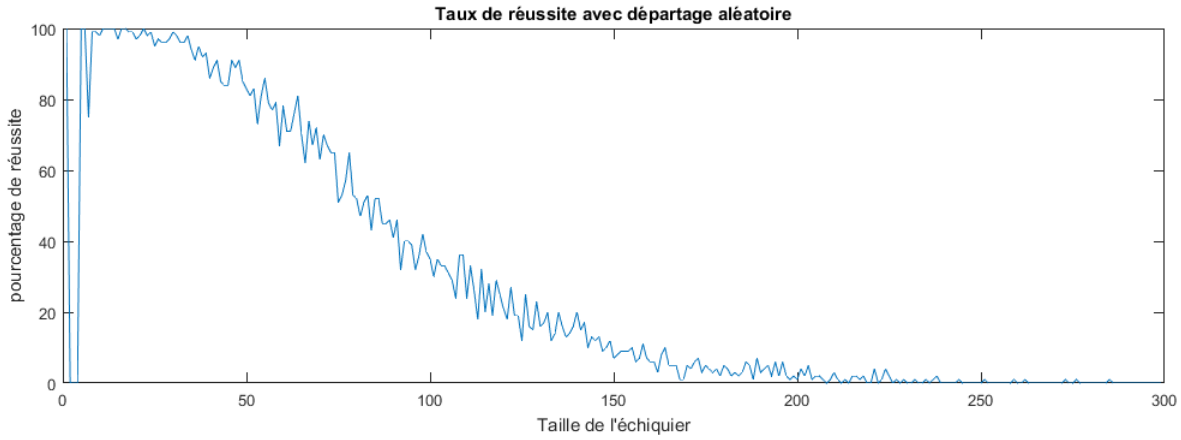

FIGURE 3 – Taux de réussite pour la génération de chemins hamiltoniens sur échiquiers carrés avec l'heurisitque de Warnsdorff

est jaune, le taux de réussite est proche de 100%. Si elle est bleue, le taux de réussite est proche de 0%. Nous pouvons observer que les résultats les plus satisfaisants sont pour les échiquiers avec m et n inférieurs à 50. De plus, plus m et n sont proches, plus les résultats sont concluants. Cette observation n'est pas surprenante si on se fie aux résultats sur les échiquiers carrés.

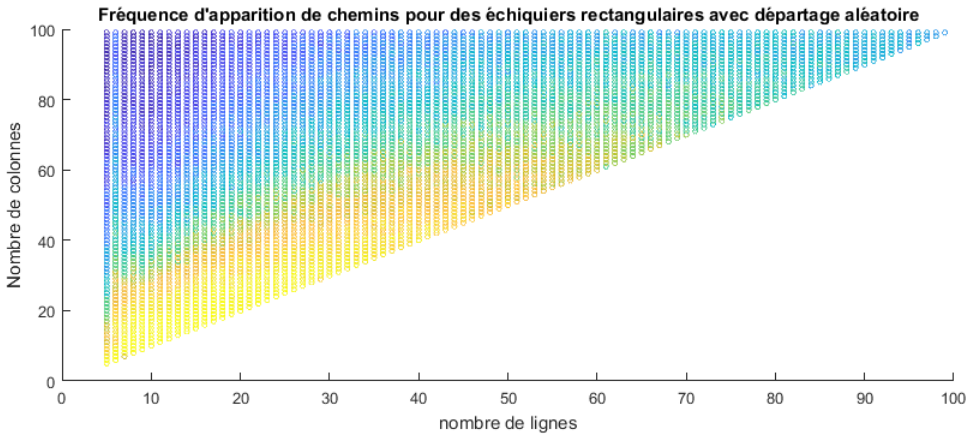

FIGURE 4 – Taux de réussite pour la génération de chemins hamiltoniens sur échiquiers rectangulaires avec l'heurisitque de Warnsdorff

Nous avons également tenté de trouver des circuits hamiltoniens avec le départage aléatoire sur des échiquiers carrés. Nous étions partis du principe que comme le circuit est une forme particulière du chemin, il y avait une probabilité non nulle de générer des circuits. La figure 5 présente ces résultats. Pour chaque dimension d'échiquier, nous avons réalisé 100 000 essais, et repis la proportions de circuits valides. Il apparait sans trop de surprise que les résultats obtenus sont très peu concluants. Pour un échiquier de 8 cases de côté, seuls 20 essais sur 100000 furent valides.

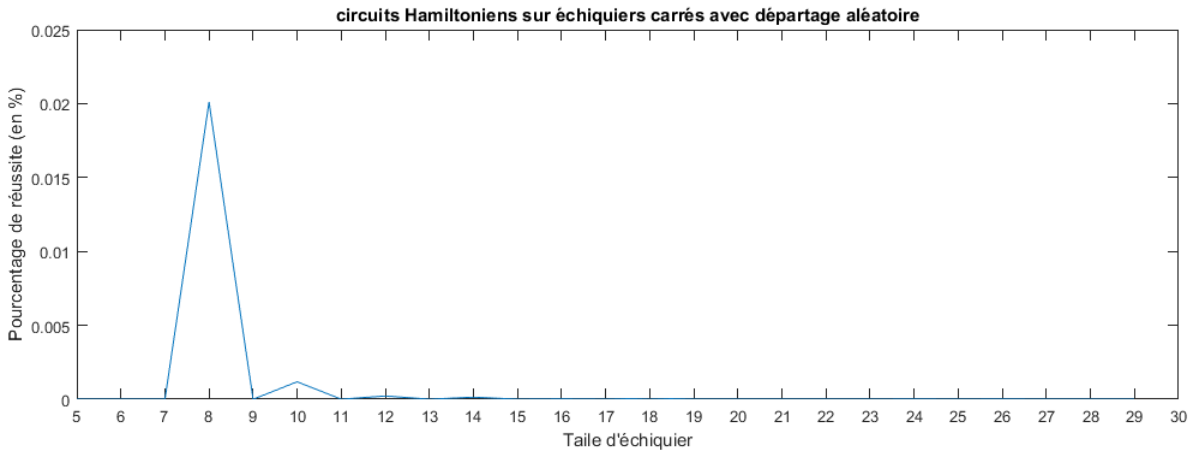

FIGURE 5 – Taux de réussite pour la génération de circuits hamiltoniens sur échiquiers carrés avec l'heurisitque de Warnsdorff

4.1.2 chemins hamiltoniens avec l'algorithme de Squirrel

Les tests effectués avec l'algorithme de Squirrel sont plus satisfaisants qu'avec le départage aléatoire. Squirrel définit son algorithme comme fonctionnel pour tout échiquier carré de plus de 112 cases de côté. Il propose également des permutations de l'ordre de parcours efficaces pour des échiquiers de taille inférieure à 112. Nous avons effectué nos tests uniquement avec l'algorithme originel. Et celui-ci s'est montré fiable à tous les coups pour tout échiquier de plus de 5 carrés de côté. A une exception, lorsqu'il y a 74 cases de côté. Dans ce cas là, l'algorithme rate à tous les coups. Il serait donc intéressant de tester une permutaiton fonctionnelle pour cette dimension d'échiquier.

La figure 6 montre les temps de calcul obtenus avec l'algorithme de Squirrel pour des échiquiers avec 5 <= n <= 10000. L'évolution des temps de calcul est légèrement supérieure à la complexité $\theta(n^2)$ annoncée. Cela est certainement dû à notre implémentation en C.

4.1.3 Algorithme de Shun-Shii

Il n'y a pas grand chose à rajouter sur l'implémentation de cet algorithme. Celui-ci trouve un chemin hamiltonien pour tout échiquier de taille 3*m. Si tant est qu'un chemin existe, conformément au théorème de Schwenk. La complexité est similaire à celle de l'algorithme de Squirrel.

5 Annexe

5.1 Compilation

La compilation se fait en utilisant cmake et make.
Depuis le répertoire source :
— mkdir build
— cd build

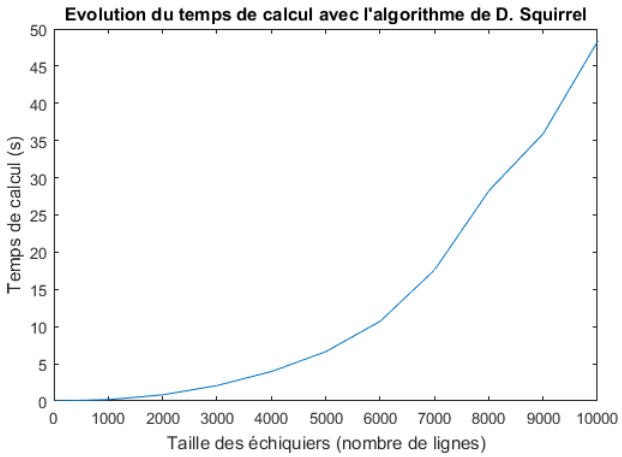

FIGURE 6 – Temps de calcul mesurés avec l'algorithme de Squirrel

— cmake ..
— make

5.2 Execution

Deux exécutables sont généres, **path** et **path_all**

path Il n'y a pas d'arguments pour path. Tout se fait via un menu.

path_all peut predre jusqu'à 3 arguments. Si ils sont mal passés l'application utilisera les arguments par défaut -o 5 5.

— Type de parcourt ouvert -o ou fermé -c
— Largeur de la grille u_int
— Hauteur de la grille u_int

5.3 Sources

— **Wikipedia : Problème du cavalier** https ://fr.wikipedia.org/wiki/Probl
— **Wikipedia : Problème du cavalier** `https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_cavalier`
— **Zanotti : Le problème du cavalier** `http://zanotti.univ-tln.fr/ALGO/I51/Cavalier.html`
— **Baylede : Problème du cavalier** `http://bayledes.free.fr/carres_magiques/Cavaliers.html`
— **Graphs and Graph Algorithms** `http://interactivepython.org/runestone/static/pythonds/Graphs/toctree.html`
— **Knight's Tour Analysis** `http://interactivepython.org/runestone/static/pythonds/Graphs/KnightsTourAnalysis.html`
— **GeeksForGeeks : Warnsdorff** `http://www.geeksforgeeks.org/warnsdorffs-algorithm-knights-tou`

— **A Warnsdorff-Rulle Algorithms for Knight's Tours on Square Chessbords***Squirrell et Cull 1996* http://math.oregonstate.edu/~math_reu/proceedings/REU_Proceedings/Proceedings1996/1996Squirrel.pdf
— **Zestedesavoir : Graphes et représentation de graphe** https://zestedesavoir.com/tutoriels/681/a-la-decouverte-des-algorithmes-de-graphe/727_bases-de-la-theorie-des-graph3352_graphes-et-representation-de-graphe/
— **Optimal algorithms for constructing knight's tours on arbitrary n*m chessboards***Shun-Shii Lin et Chung-Liang Wei 2005* http://www.sciencedirect.com/science/article/pii/S0166218X04003488