Quadrature nº 82 (2011) 1-5

Les dominos de Wang

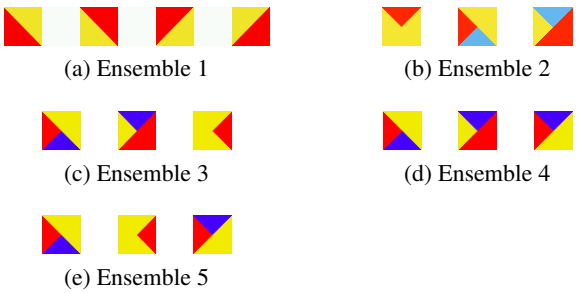
par Jean-Jacques Lévy*

Résumé.

Le pavage de rectangles par des dominos de Wang est un problème de programmation et d'algorithmique faisant intervenir beaucoup de notions d'informatique. Si son utilité pratique est vraiment secondaire, il est typique des problèmes qu'on peut traiter par exhaustivité avec la puissance de calcul des ordinateurs modernes, à condition de réfléchir un tout petit peu en termes de complexité algorithmique. Le problème du pavage du plan fait intervenir des notions de calculabilité et a joliment abouti à l'existence de pavages apériodiques, tels que les fameux diagrammes de Penrose, qui non moins curieusement se retrouvent dans des structures cristallines de la nature.

I Introduction

Les dominos de Wang sont des carrés à bords colorés comme ceux figurant dans la figure 1. Ce sont des dominos à quatre bords qui fonctionnent dans le plan à deux dimensions. On cherche à remplir des rectangles avec ces dominos, par exemple comme sur la figure 2 avec le premier ensemble de dominos jaunes et rouges de la figure 1a (chaque motif peut être utilisé un nombre arbitraire de fois). Pour des carrés de côté 6 avec ces



(a) Ensemble 1 (b) Ensemble 2

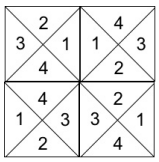
(c) Ensemble 3 (d) Ensemble 4

(e) Ensemble 5

Figure 1: Exemples de motifs

quatre types de dominos, il existe 4 096 pavages différents. Un pavage est licite si deux dominos adjacents ont leur côté commun de même couleur. Il est interdit de tourner les motifs, ce qui ne fait pas de différence dans cet exemple. Mais cela compte dans l'exemple suivant avec les trois dominos de la figure 1b. Pour les carrés de côté 6, il n'existe alors que les 3 pavages représentés sur la figure 3. Mais il est impossible de paver ces carrés de côté 6 avec les motifs de la figure 1c ou 1d, alors qu'on peut le faire avec ceux de la figure 1e !

Les dominos sont donc formellement définis comme des quadruplets (d,h,g,b) d'entiers donnant les couleurs des côtés droit, haut, gauche et bas. Le problème des dominos de Wang s'énonce comme suit : étant donné p motifs et un carré de côté n, existe-t-il un pavage du carré avec ces motifs (nous énonçons le problème avec des carrés à remplir, plutôt qu'avec des rectangles pour simplifier, mais l'essence du problème est la même avec des rectangles) ?



On remarque que le problème devient trivial si les rotations de motifs sont permises, puisqu'on peut construire un motif périodique avec un seul motif et des rotations de 180 degrés, comme indiqué sur la figure ci-contre. Les rotations sont donc interdites. Nous allons considérer le problème en deux temps : le pavage de carrés de côté n fini et le pavage du plan ($n = \infty$).

* Jean-Jacques.Levy@inria.fr

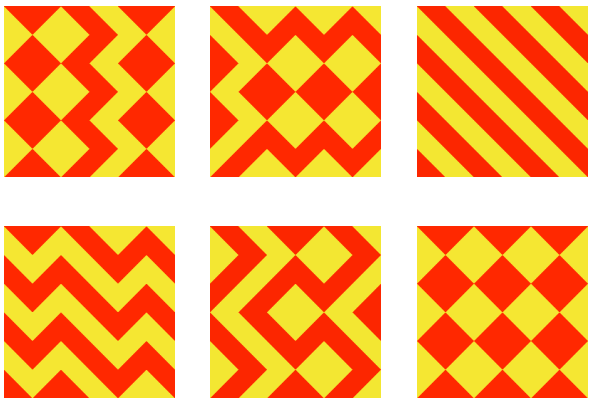


Figure 2: Pavages de carrés de côté 6 avec les dominos de l'ensemble 1

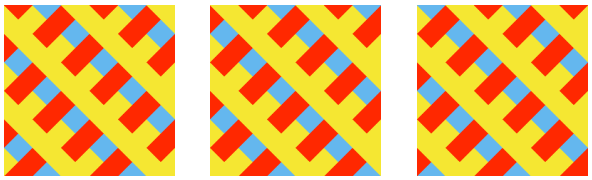


Figure 3: Pavages de carrés de côté 6 avec les dominos de l'ensemble 2

II Pavages finis

Comment trouver un bon pavage par programme ? Une première manière consiste à énumérer tous les pavages pour ne conserver que ceux qui sont autorisés. Ce programme s'écrirait comme suit en Caml :

```
let rec paver(carre, i, j, motifs) =
  let n = largeur(carre) in
  if i >= n then begin
    if estLicite(carre) then
       incr nbSolutions;
  end else
    foreach c in motifs do
      carre.(i).(j) <- c;
      if j+1 >= n then
        paver(carre, i+1, 0, motifs)
      else
        paver(carre, i, j+1, motifs) ;
    done;;
```

Tous les pavages sont trouvés en appelant la fonction `paver` avec $i = j = 0$. Le nombre de bons pavages est donné par la variable `nbSolutions`. Au début on a `nbSolutions` $= 0$. La variable `carre` est un tableau représentant une matrice $n \times n$ de dominos où $n =$ `largeur(carre)`. La fonction `estLicite` teste si le pavage est correct, et si c'est le cas, on incrémente le nombre de solutions trouvées. La variable `motifs` représente un ensemble de dominos, et la pseudo-instruction `foreach` fait une itération sur tous les éléments de cet ensemble (On pourra par exemple le représenter par une liste et dans ce cas, la variable c prendra successivement la valeur de tous les éléments de cette liste). On remarque enfin que la fonction `paver` est élégamment récursive et qu'il est pratiquement impossible de l'écrire sans récursivité. Cette fonction part de la case (i, j) et trouve tous les pavages possibles à partir de cette case. On essaie un premier motif c de `motifs` sur cette case et on recommence récursivement avec les pavages démarrant à partir de la case suivante, jusqu'à remplir tout le carré. Arrivé au bout du carré, on teste la validité du pavage, auquel cas on augmente le nombre de solutions. Puis on fait un retour en arrière en essayant d'autres motifs sur la case (i, j) et on repart en avant à nouveau récursivement sur la case suivante, etc. Ce style de programmation, *backtracking* en anglais, est très courant dans l'exploration systématique d'un espace arborescent de solutions.

La fonction `paver` énumère les p^{n^2} pavages du carré pour ne retenir que ceux qui sont autorisés. Pour 3 motifs et un carré de côté 6, cela représente 150994635296999121 pavages, soit de l'ordre de 10^{17} opérations, soit une trentaine d'années avec une machine moderne. Cette fonction est donc inutilisable ! On l'améliore grandement en éliminant les mauvaises solutions au fur et à mesure de leurs constructions. Cette autre technique de programmation, dénommée cette fois-ci *branch and bound* en anglais, est souvent associée au *backtracking*. Ici on ferait le test `okBords` de légalité de la solution quand on introduit un nouveau domino en position (i, j) au lieu d'attendre la fin du pavage de tout le carré. Cette fonction teste si les bords gauche et haut du domino placé en position (i, j) ont une même couleur que les bords droit et bas des dominos en positions $(i, j-1)$ et $(i-1, j)$ respectivement. Ceci donne un nouvelle version de la fonction `paver`.

```
let rec paver(carre, i, j, motifs)  =
  let n = largeur(carre) in
  if i >= n then
      incr nbSolutions;
  else
    foreach c in motifs do
      carre.(i).(j) <- c;
      if okBords(carre, i, j) then
        if j+1 >= n then
          paver(carre, i+1, 0, motifs)
        else
          paver(carre, i, j+1, motifs) ;
    done;;
```

En moyenne, cette solution marche très bien et permet d'obtenir rapidement une solution, tout en étant aussi mauvaise que la précédente dans des cas extrêmes. Dans le pire cas, ce programme a toujours une complexité exponentielle en $O(p^{n^2})$ pour p motifs et un carré de côté n.

Sait-on faire mieux ? Existe-t-il une solution qui ne soit pas exponentielle par rapport à la surface du carré à remplir ? En fait, personne ne le sait car le problème

du pavage d'un carré par des dominos de Wang est *NP*-complet. Les problèmes *NP*-complets ont été définis par Stephen Cook en 1971 et étudiés par Richard Karp en 1973. C'est la classe la plus générale de problèmes traitables en temps polynomial par un algorithme non-déterministe. Dans notre exemple, on sait résoudre le pavage en temps polynomial en faisant appel à un oracle non-déterministe. On reprend le programme précédent, mais au lieu de considérer tous les motifs, à chaque nouvel emplacement (i, j) dans le carré, l'oracle sélectionne un motif ou dit d'arrêter s'il n'y a pas de solution. Mais, comme l'oracle peut se tromper, à la fin du pavage, on vérifie en temps polynomial (ici en temps $O(n^2)$, donc linéaire par rapport à la surface du carré) que la solution obtenue avec l'aide de l'oracle est bien autorisée. On obtient donc une solution non-déterministe en temps linéaire, et on sait tester la solution en temps également linéaire. Le pavage par dominos de Wang est clairement résoluble par un algorithme *NP* (non-déterministe polynomial). Pour montrer que ce problème est *NP*-complet, c'est-à-dire le plus général dans la classe *NP*, il faut montrer qu'un autre problème *NP*-complet, par exemple le problème 3SAT de satisfiabilité des formes normales conjonctives de trois littéraux, peut se coder en temps polynomial avec nos dominos, ce que nous ferons dans la section IV. Donc, si on sait trouver une meilleure solution au problème des dominos de Wang, on saurait également trouver une meilleure solution pour tous les problèmes de la classe *NP*.

Or savoir si $P = NP$, c'est-à-dire si la classe P des problèmes résolubles par des algorithmes déterministes en temps polynomial coïncide avec la classe des problèmes *NP*, est une des questions non résolues les plus intrigantes de l'informatique. Cook a posé cette question en 1971, des générations d'informaticiens théoriciens ont essayé de trouver la réponse. Des mathématiciens prestigieux s'y sont attaqués sans succès. Cette conjecture fait maintenant partie des 7 problèmes du millénaire, définis en 2000 par l'institut Clay de mathématiques, avec une récompense d'un million de dollars à la clé pour chacun d'entre eux. Il est probable que $P \neq NP$, mais personne ne sait le démontrer !

III Pavages du plan

À présent, on s'intéresse au pavage du plan avec p motifs ($n = \infty$). Clairement, si on sait paver tout le plan, on sait paver tous les carrés de côté fini n. La réciproque est également vraie, mais demande un peu plus de raisonnement. On considère l'arbre dont les nœuds sont les pavages licites de tous les carrés de taille finie. La racine de l'arbre est le pavage du carré de côté nul. Un nœud représentant un pavage d'un carré de côté n admet comme fils dans l'arbre tout nœud représentant un pavage d'un carré de côté $n+1$ qui est une extension du pavage de son père. Comme le nombre p de motifs est fini, chaque nœud n'a qu'un nombre fini de fils. Donc, cet arbre qui a un nombre infini de nœuds (puisqu'on sait paver tous les carrés de taille n), admet un chemin infini (par le lemme de König). On peut donc paver tout le plan avec ces p motifs [1].

Hao Wang a conjecturé en 1961 que tout pavage du plan avec un nombre fini de motifs, est périodique. Il existe donc un algorithme qui trouve le pavage du plan avec ces motifs et répond non si c'est impossible. Cet algorithme lance en parallèle les deux semi-algorithmes suivants : trouver le motif périodique et chercher un contre-exemple. Plus précisément, on considère successivement tous les carrés de côté $0, 1, 2, \ldots\ n, n+1$, etc. Soit on trouve un carré impossible à paver et il est impossible de paver tout le plan, soit on finit par trouver un motif périodique et on peut utiliser ce motif périodique pour paver tout le plan. Cet algorithme décide donc en fonction des motifs si le plan est pavable.

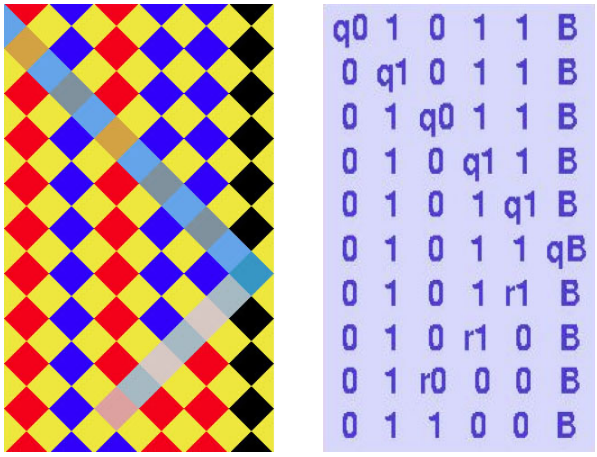


Figure 4: Calcul du successeur : $11 = 8+2+1$ (1011 en binaire) sur la ligne du haut, $12 = 8+4$ (1100 en binaire) sur la ligne du bas. Le bit 1 est bleu, le bit 0 est rouge, le marqueur B de fin est noir.

En 1966, Robert Berger a montré que cet algorithme est impossible par un argument de la théorie de la calculabilité de Kleene. En effet, on peut montrer que toute fonction calculable peut se calculer avec les dominos de Wang. Prenons le cas simple de la fonction $x \mapsto x+1$ qui retourne le successeur $x+1$ de tout entier naturel x. Il existe 14 motifs tels que, pour tout entier x écrit en binaire suivi du marqueur de fin B, le rectangle de largeur $\ell+2$ et de longueur $\ell+\ell'+4$ (où ℓ est le nombre de chiffres de x en binaire et ℓ' est le

1. Le lemme de König dit que tout arbre qui a un nombre infini de nœuds et dont tout nœud a un degré fini admet un chemin infini.

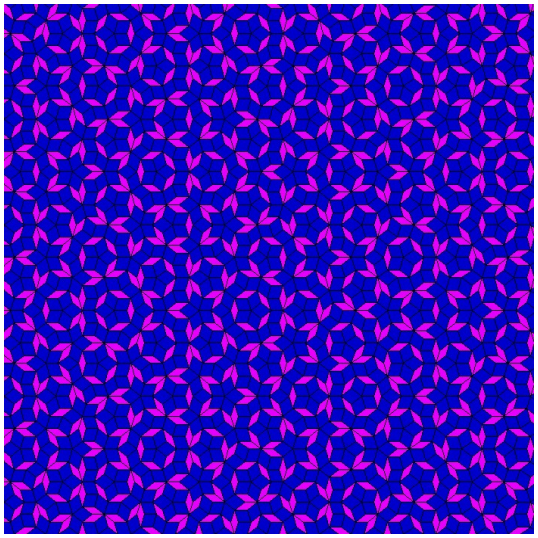


Figure 5: Pavage de Penrose

nombre de 1 consécutifs à droite de x en binaire) est pavable avec la première ligne contenant le nombre x et la dernière $x+1$, comme le montre l'exemple de la figure 4. Dans cet exemple, le chiffre 0 correspond à un bord rouge, et 1 correspond à un bord bleu. Le calcul démarre dans l'état q_0 et on part vers la droite en alternant entre les états q_0 et q_1 selon qu'on rencontre un 0 ou un 1 en attendant le marqueur de fin B (noir) où on se met dans un état q_B. On repart alors vers la gauche en ajoutant 1, dans un état r_1 qui propage la retenue jusqu'au premier 0 qui donne un état de fin r_0. Par ailleurs, pour imposer de démarrer avec la ligne q_0, suivi de plusieurs 0 ou 1 et du marqueur de fin B, il faut quelques motifs supplémentaires avec des couleurs uniques pour les bords supérieur, gauche pour q_0 ou droit pour B. De même pour imposer que la dernière ligne soit un nombre suivi du marqueur B, on considère des motifs avec cette fois-ci un bord bas de couleur unique (ou droit pour le marqueur). Ainsi ce rectangle est pavé si et seulement si la dernière ligne est obtenue à partir de la première en ajoutant 1. On peut de même coder l'addition sur les entiers, puis la multiplication, l'exponentiation et toutes les fonctions calculables. En fait, Berger a montré que toute machine de Turing pouvait s'exprimer avec de tels pavages. Donc si on sait décider du pavage des rectangles par des dominos de Wang, on sait décider si $y = f(x)$ pour tous entiers x et y et fonction calculable f. Or la théorie de Kleene dit qu'il n'existe pas de tel algorithme.

La conjecture de Wang est donc fausse. Par un argument de la théorie de la calculabilité, on démontre donc qu'il existe des pavages apériodiques du plan ! Mais quels sont-ils ? Dans les années 1970-1980, les pavages apériodiques ont fleuri, au début avec 20 426 motifs par Berger, puis réduits à 104 par Berger, et 13 en 1996 par Culik ! Penrose obtint des pavages apériodiques plus simples en changeant les règles du problème avec des motifs de géométrie différente : triangles, fléchettes ou losanges. Les pavages de Penrose s'obtiennent par des symétries de $\pi/5$, ils sont quasi périodiques, puisque toute zone rectangulaire se retrouve régulièrement ailleurs. Ce jeu mathématique a trouvé une utilité étonnante en chimie, puisque certaines structures cristallines, des quasi-cristaux, ont été montrées comme étant apériodiques.

IV NP-complétude

Montrons à présent que le pavage de rectangles (de taille finie) est *NP*-complet. On part du problème 3SAT et on montrera qu'on peut le réduire en temps polynomial au pavage par des dominos de Wang. Le problème 3SAT considère des formules logiques de la forme

$$(x_1 \vee \neg x_3 \vee x_5) \wedge (\neg x_2 \vee x_3 \vee x_4) \\ \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_5)$$

et consiste à trouver des valeurs booléennes pour les variables x_1, x_2, x_3, x_4, x_5 rendant la formule vraie. On dit alors que la formule est satisfiable. Le problème s'appelle 3SAT car la formule à satisfaire est une suite de conjonctions de clauses disjonctives d'exactement trois littéraux (un littéral est une variable ou son complément). Dans notre exemple, la valuation $x_1 = x_5 = \texttt{vrai}$, $x_2 = x_3 = x_4 = \texttt{faux}$ satisfait la formule précédente.

Le codage de 3SAT dans les dominos de Wang ressemble à l'exemple du calcul de la fonction successeur de la section précédente. La formule à satisfaire est cette fois-ci placée sur la colonne de gauche, comme sur la figure 6. Les n variables booléennes $x_1, x_2, \ldots x_n$ sont sur la ligne du haut à partir de la troisième colonne. On cherche à paver le rectangle avec des dominos représentant une valuation possible de ces n variables qui rend vraie la formule de la colonne de gauche. Pour coder la valuation des n variables, la ligne du haut contient des dominos de la forme $(\texttt{v}_i, \texttt{n}, \texttt{v}_i, \texttt{n})$, $(\texttt{f}_i, \texttt{n}, \texttt{f}_i, \texttt{n})$ où $\texttt{n}$ est une couleur neutre, $\texttt{v}_i$ et $\texttt{f}_i$ sont des couleurs toutes distinctes représentant les valeurs $\texttt{vrai}$ et $\texttt{faux}$ pour les variables x_i ($1 \leq i \leq n$). La colonne de gauche contient pour chaque littéral la valeur de la variable correspondante qui le rend vrai. Cette colonne contient donc des dominos de la forme $(\texttt{n}', \texttt{v}_i, \texttt{n}', \texttt{v}_i)$ pour le littéral x_i ou $(\texttt{n}', \texttt{f}_i, \texttt{n}', \texttt{f}_i)$ pour le littéral $\neg x_i$ (où $\texttt{n}'$ est une autre couleur neutre). La deuxième colonne évalue la clause et produit le résultat $\texttt{V}$ ou $\texttt{F}$ dans la case du bas. Il y a deux types de dominos dans cette deuxième colonne. Pour chaque littéral, il y a un domino de la forme (b, g, h, d) tel que b est le résultat de l'expression booléenne $h \vee (g = d)$. Entre deux clauses, il y a un

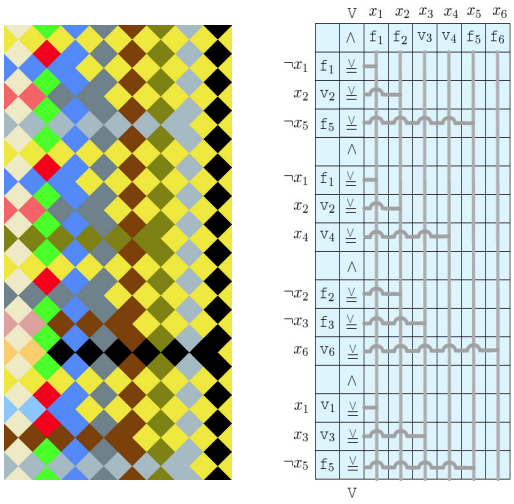


Figure 6: Satisfiabilité de la formule $(\neg x_1 \vee x_2 \vee \neg x_5) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_6) \wedge (x_1 \vee x_3 \vee \neg x_5)$ avec la valuation $x_1 = x_2 = x_5 = \texttt{faux}$ et $x_3 = x_4 = \texttt{vrai}$. Sur la deuxième colonne, la couleur verte correspond à la valeur `vrai`, le rouge est pour `faux`. L'évaluation de la clause se fait de haut en bas, avec les dominos (b,g,h,d) vérifiant $b \equiv h \vee (g = d)$. La valeur finale se trouve en bas de la deuxième colonne. Les couleurs jaune et jaune pâle sont neutres.

domino $(\texttt{F},\texttt{n},\texttt{V},\texttt{n})$ qui s'assure que la clause du dessus est vérifiée et initialise à faux le calcul de la clause du dessous. La formule entière est vérifiée si on démarre sur la première ligne avec la valeur faux et si on obtient la valeur vraie sur la dernière ligne. Il ne reste plus qu'à assurer la connexion entre les valeurs données aux variables x_i et les valeurs demandées pour évaluer chaque littéral. Ce n'est plus qu'un codage de la circuiterie nécessaire pour connecter ces valeurs sur la ligne du haut vers les valeurs nécessaires sur la deuxième colonne. Ceci est réalisé dans les cases figurant sous la ligne des variables et à droite de la deuxième colonne par des dominos de la forme $(\texttt{n},\texttt{v}_i,\texttt{v}_i,\texttt{v}_i)$, $(\texttt{n},\texttt{f}_i,\texttt{f}_i,\texttt{f}_i)$ et (d,h,d,h) pour $h \in \{\texttt{v}_i,\texttt{f}_i\}$ et $d \in \{\texttt{v}_j,\texttt{f}_j\}$ $(i \neq j$ et $1 \leq i \leq n)$. Ainsi la surface nécessaire n'est pas plus grande que $(2+n) \times (p+2)$ où n est le nombre de variables, p la longueur de la formule à satisfaire (il faut quelques dominos supplémentaires si on veut assurer l'unicité du pavage).

Ainsi le problème 3SAT se réduit au problème du pavage de rectangles par des dominos de Wang. Ce dernier problème est donc *NP*-complet.

V Conclusion

Avec les dominos de Wang, le pavage du plan est indécidable et le pavage d'un rectangle ou carré est *NP*-complet. Ces deux problèmes a priori anodins sont donc soit impossibles, soit difficiles à résoudre par programme. Pourtant, beaucoup de problèmes sont traités algorithmiquement. D'abord un bon nombre de problèmes *NP*-complets sont résolus en pratique en faisant un certain nombre d'approximations, ou d'heuristiques adéquates. Par exemple, avec nos dominos, la solution par backtracking donne en général de bons résultats. Mais la plupart des problèmes ne sont pas *NP*-complets et ont une complexité beaucoup plus faible, linéaire, quadratique ou au pire cubique dans la taille de leurs données. On arrive ainsi à construire les nouvelles cathédrales des temps modernes que sont les grands systèmes informatiques.

Références

[1] WANG, H. *Proving theorems by pattern recognition—II*, Bell System Tech. Journal 40(1) :1–41. (Wang propose ses dominos, et conjecture la périodicité du pavage du plan), 1961

[2] BERGER, R. (1966). *The undecidability of the domino problem*, Memoirs Amer. Math. Soc. 66. (Utilise le mot *Wang tiles*, et contredit la conjecture de Wang), 1966

[3] GAREY, M. R. et JOHNSON, D. S., *Computers and Intractability : A Guide to the Theory of NP-Completeness*, W. H. Freeman, ISBN 0-7167-1045-5 (La référence sur les problèmes *NP*-complets), 1979

[4] HOPCROFT, J. E., MOTWANI R. et ULLMAN, J. D., *Introduction to Automata Theory, Languages, and Computation*, Prentice Hall, ISBN-10 : 0321462254 (Très bon livre sur les modèles de complexité), 2007

[5] KÖNIG, D., *Sur les correspondances multivoques des ensembles*. Fundamenta Mathematicae, (8) : pp. 114–134 (Le lemme de König), 1926.

[6] HAREL, D., *Algorithmics : The Spirit of Computing*, 3rd edition, Addison-Wesley, ISBN 10 : 0321117840 (Introduction aux algorithmes), 2004.

David Harel, professeur à l'institut Weizmann a popularisé le problème des dominos de Wang dans son excellent livre *Algorithmics : The Spirit of Computing* qui sont les notes d'un cours donné d'octobre 1984 à janvier 1985 sur la radio israélienne *Galei Zahal*.

Une version plus colorée de cet article figure sur le *web* en `http://jeanjacqueslevy.net/pubs/11quadrature.pdf`