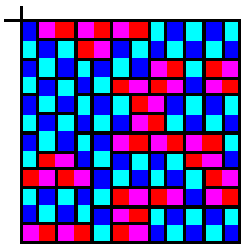
Arbres Couvrants Aléatoires
Pavages par Dominos

Mémoire



Thibault Godin
encadrant : M. Pol Vanhaecke

Université de Poitiers
Département de Mathématiques
2012-2013

Table des matières

Introduction

Bien qu'ils paraissent simples, les pavages par dominos sont une branche intéressante des mathématiques, aux croisement de plusieurs sujets, notamment les mathématiques discrètes, la géométrie et le combinatoire. L'ajout d'aléas mène à des modèlisations physiques, entre autres le modèle des dimères et la percolation, datant de la fin des années 50 et qui est une branche active de la recherche mathématique. Mais même sans cela des structure intéressantes commes les groupes, les treillis ou des problèmes mettant en œuvres des solutions rafinées apparaissent [4]. Ces pavages sont sensibles à plusieurs paramètres, par exemple la forme : un pavage du carré a un comportement limite bien différent d'un pavage du diamant aztèque ($D_n = \{x, y \in \mathbf{Z}, |x - 1/2| + |y - 1/2| \leq n\}$, pour plus de détails voir [3]). Cependant trouver un pavage même par des dominos et dans le cas simple d'un rectangle pavable non troué s'avère complexe si l'on s'en tient à l'approche naïve. C'est pourquoi il a été necessaire de trouver des liens entre ces pavages et des objets plus simples et mieux connus. L'utilisation de la théorie des graphes s'est avérée pertinente car on peut représenter un pavage comme un couplage parfait d'un graphe, c'est la bijection de *Temperley*. Cependant le problème de couplage reste complexe, et l'on peut sauter cette étape en exprimant une bijection directement entre le pavage du rectangle et l'arbre couvrant (on précisera quel graphe exactement par la suite). C'est cette adaptation que l'on va présenter dans ce mémoire et surtout on va en donner une preuve complète. De plus cette bijection étant très algorithmique on peut la mettre en œuvre informatiquement. On a donc programmé en `Scilab` un algorithme générant le graphe mis en jeu par la bijection puis un algorithme qui en extrait un arbre couvrant et enfin un programme dessinant le pavage à partir de l'arbre donné. On arrive donc à créer assez simplement des pavages valables de façon aléatoire, comme on le voit sur la page de garde.

De plus cette bijection permet de généraliser des notions de théorie des graphes aux pavages réciproquement, mais ausssi d'utiliser des résultats déjà démontrés sur un type d'objet. Par exemple dénombrer les pavages via le nombre d'arbres couvrants en se servant théorème Matrix-Tree de *Kirchhoff*, alors qu'une approche directe est compliquée en dehors de cas très particuliers, par exemple le cas $2 \times n$ où le nombre de pavages est le n-ième nombre de *Fibonacci*.

On decouvre aussi des liens avec les chaînes de *Markov* par des algorithmes de flip ou l'algorithme de *Wilson* et les marches à boucles effacées qui pourraient faire l'objet d'approfondissements ultérieurs. D'autre part on peut aussi s'intéresser à d'autre forme de pavé élémentaire (losange, hexagone) ou de surface (diamant aztèque, polymino) et chercher des liens entres eux. Enfin l'étude des comportements limites et de leurs traductions physiques fournit un vaste champs d'exploration pour de futurs projets.

Les programmes sont disponibles sur demande par mail à `thibault.godin@etu.univ-poitiers.fr`.

Pour ce mémoire je tiens à remercier sincèrement M. Pol Vanhaecke pour son aide, sa gentillesse et sa disponibilité.

1 Définitions

Définition 1 *Un* graphe *G est un ensemble de sommets V et d'arêtes E muni d'une application $e : E \to S$ avec $S = \{e(a) \in \mathcal{P}(E), card(e(a)) \in \{1, 2\}\}$*

Remarque : $e(a)$ donne les extrémités de l'arête a (donc un sommet pour une boucle ou deux sommets différents). Le concept d'arête est donc ici non orienté.
On peut attribuer un poids aux arêtes. Le graphe est alors dit pondéré ou valué et on note $w(a)$ le poids de l'arête a (formellement on munit E d'une application w vers $\mathbf{R}$).

Définition 2 *Une* chaîne *entre deux sommets u et v de V est une suite non vide $a_1, \ldots, a_n$ d'arêtes telle que $u \in e(a_1); v \in e(a_n)$ et qu'il existe une suite $u_1 \ldots u_{n-1}$ de sommets avec $u_i \in e(a_i) \cap e(a_i + 1),\ i \in [|1, n-2|]$.*

Définition 3 *On dit qu'un graphe est* simple *si $\forall a \in E, card(e(a)) = 2$ et $\not\exists a_1, a_2 \in E, a_1 \neq a_2, e(a_1) = e(a_2)$. Donc que chaque arête relie deux sommets distincts et qu'entre deux sommets il passe au plus une arête. Un* cycle *est une chaîne simple telle que $u = v$.*

Remarque : quand le graphe est simple on notera $uv = vu$ l'arête a telle que $e(v) = (u, v)$. Alors un chaîne entre u et v sera une suite $u, u_1, u_1u_2, \ldots, u_{k-1}u_k, u_kv$.

Définition 4 *Un graphe est* connexe *s'il existe un chaîne entre tout couple de sommets.*

Définition 5 *Une* forêt *est un graphe sans circuit*

Définition 6 *Un* arbre *est un graphe connexe sans circuit*

Définition 7 *Un arbre $T = (V_T, E_T)$ (resp. une forêt $F = (V_F, E_F)$) est dit couvrant pour le graphe $G = (V, E)$ si $V_T = V$ et $E_T \subset E$ (resp. $V_F = V$ et $E_F \subset E$).*

Définition 8 *Le* poids *d'un graphe G est la somme du poids de ses arêtes. On le notera $w(G)$.*

On a alors une relation d'ordre sur les sous-graphes d'un graphe pondéré.

Pour un graphe quelconque $G = (V, E)$ on notera n le nombre de sommets ($n = |V|$), m le nombre d'arêtes ($m = |E|$) et on supposera ces quantités finies.

À partir de maintenant on se place sur un rectangle de $\mathbf{R}^2$ à coordonnées entières, colorié à la manière d'un échiquier.

Définition 9 *Un domino est un rectangle d'aire 2 de $\boldsymbol{R}^2$ à coordonnées entières. Chaque domino possède une case sombre, qui correspond à la case noire de l'échiquier et une case claire.*

On se donne une orientation, du clair au sombre, pour nos dominos.
On définit la notion de cycle de dominos :

Définition 10 *Étant donné un domino (à gauche dans le dessin), un domino bien posé par rapport à lui est un domino qui respecte l'un des trois schémas suivants (à rotation de $k\frac{\pi}{2}$ près) :*

s r l

Une suite de dominos bien posés est un ensemble $(d_n)_{n\in I\subset \mathbf{N}}$ telle que $\forall i \in I, d_{i+1}$ soit bien posé par rapport à d_i et qu'aucun dominos se chevauchent ($\forall i,j \in I, d_i \cap d_j = \emptyset$). Pour chaque suite de dominos bien posés l'orientation induit une relation père/fils : dans une suite de dominos bien posés le père d'un domino est le domino qui le précede immédiatement, il est unique et existe toujours, sauf éventuellement pour un domino d_0 qui sera alors appelé domino d'origine ou départ. Le fils est, en cas d'existence, un domino qui lui succède. Un domino peut avoir 0 ou 1 fils, le nombre de fils sera appelé son degré. Un domino sans fils sera appelé bout ou extrémité.

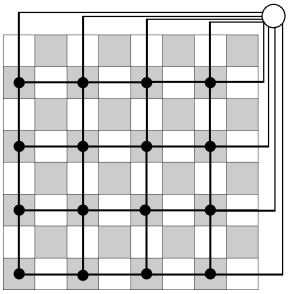
Définition 11 *Une suite D de dominos est un cycle si $I = [|0;k|], k \in \mathbf{N}^*$ et d_0 est bien posé par rapport à d_k*

Enfin on rappelle la définition de l'étoile de Kleene :

Définition 12 *Soit $X = \{x_1, x_2, \ldots\}$ un alphabet, on définit $X^n = \{a_1a_2\ldots a_n | \forall i \in [|1;n|], a_i \in X\}$ avec comme convention $X^0 = \epsilon$ où ϵ est le mot vide. La fermeture de Kleene, aussi appelée étoile de Kleene est la réunion de toutes les puissances de l'alphabet : $X^* = \bigcup_{n\in\mathbf{N}} X^n$.*

2 Bijection de Temperley

Dans [1] *H. N. Temperley* décrit une bijection entre les pavages aléatoires par des dominos horizontaux ou verticaux 2×1 et les couplages parfaits du graphe grille. On va traduire cette bijection aux notions de pavage et d'arbre couvrant d'un graphe grille.
On se place dans un rectangle $2n \times m$ colorié à la manière d'un échiquier, pour nous repérer on notera (i,j) le carré dont le sommet gauche-bas est en coordonnées (i,j). On suppose que le carré $(0,0)$ est colorié en noir. On peut alors construire des graphes en utilisant des sous ensembles de carrés comme sommets. Pour notre bijection on s'interessera aux sommets noirs et l'on considèrera qu'il existe une arête entre deux sommets s'ils sont exactement à une distance de 2 pour la distance $d((i,j);(k,l)) = \sqrt{|i-k|^2 + |j-l|^2}$. Le graphe a alors deux composantes connexes, l'une formée des sommets de type $(2k, 2l)$ (sommets noirs "pairs"), l'autre des sommets de forme $(2k+1, 2l+1)$ (sommets noirs "impairs"). À chacun des sous-graphes on rajoute un sommet extérieur, dont les voisins sont les sommets noirs respectivement pairs ou impairs situés à une distance inférieure à 2 d'un bord. Le sommet du coin possède deux arêtes vers ce sommet extérieur. On notera G_p et G_i ces deux sous-graphes.



G_p pour un carré 8×8

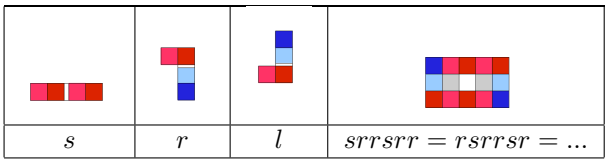
Une reformulation de la bijection de *Temperley* affirme qu'il existe une correspondance entre les arbres couvrants de G_p (respectivement de G_i) et les pavages par dominos du rectangle.

On va commencer par étudier le passage d'un pavage du rectangle à l'arbre couvrant de G_p.
On construit l'arbre en partant du sommet extérieur. On ajoute des sommets au fur et à mesure si l'on peut passer d'un sommet déjà dans l'arbre à un autre en ne traversant qu'un bord de domino. Ainsi l'arbre T_p est forcément un sous graphe de G_p, puisque deux sommets non reliés dans G_p sont distants de plus de 2 et donc qu'il faut traverser au moins deux bords de domino. Il s'agit bien d'un arbre d'après le lemme suivant :

Lemme 1 *Dans un pavage on ne peut pas avoir un cycle de dominos couvrants uniquement des sommets de même parité.*

démonstration : On va raisonner par l'absurde et montrer que l'intérieur d'un cycle est formé par un nombre impair de cases et n'est donc pas pavable.

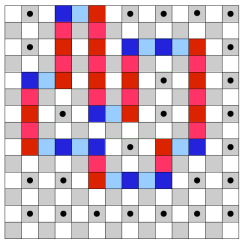
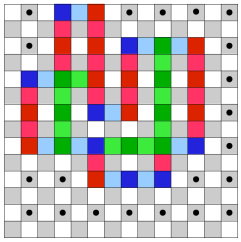
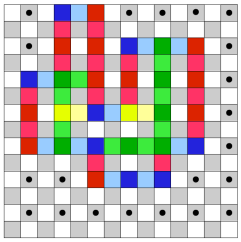
Soit un cycle de domino. Ce cycle sépare le plan en deux composantes 8-connexes, on va montrer que l'intérieur est constitué d'un nombre impair de cases. On va coder la boucle sous la forme de mots de type $(s^*r^*l^*)^*$ avec s une pose de domino dans le même sens, r une pose de domino vers la droite (par rapport au sommet précédent) et l une pose de domino vers la gauche. Comme le cycle suit les règles de pose des dominos on obtient un codage (non unique puisque le choix de l'origine est arbitraire) du cycle de dominos. On considérera ces mots à permutation circulaire près.



On va alors effectuer des simplifications dans la boucle avec comme objectif d'arriver à un rectangle de dominos sans pour autant changer la parité du nombre de cases intérieures à la boucle.
Nécessairement on a soit quatre r de plus que de l soit quatre l de plus que de r pour qu'on ait une boucle (on le prouve en annexe A). Mettons qu'il y ait plus de r.

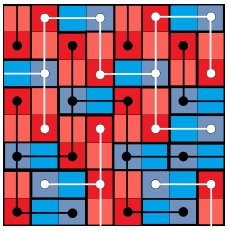
On se donne alors des règles de simplification par suppression de motif :
1 $\dots ls^irs^jrs^il\dots \to \dots s^j\dots$
2 $\dots s^irs^jrs^{i-1}l\dots \to \dots rs^{j-1}\dots$
2 bis $\dots ls^irs^jrs^{i+1}l\dots \to \dots s^{j-1}r\dots$

Les transformations laissent $|r|-|l|$ invariant et ne rompent pas le cycle mais diminuent le nombre de l. On applique cette transformtion au triplé (l, r_1, r_2) du cycle qui minimise $\mu = min_E(d_m(l,r_1) + d_m(l,r_2), d_m(l,r_1) + d_m(r_1,r_2))$ où d_m est la distance naturelle sur les mots circulaires et $E = \{rs^*ls^*r\} \cup \{rs^*rs^*l\} \cup \{rs^*rs^*l\}$ est l'ensemble des l encadrés, précédés ou suivis de deux r. Cet ensemble n'est jamais vide s'il reste un l car le mot est vu de façon circulaire et qu'il y a quatre r de plus que de l.
Donc à chaque transformation on supprime un l. Comme ceux-ci sont en nombre fini on obtient, après un nombre fini de transformations, un motif $s^krs^lrs^krs^l$. Or ce rectangle encadre un nombre impair de cases élémentaires (voir le lemme en annexe) et donc la région n'est pas pavable. Il reste à montrer que les simplifications ne changent pas la parité du nombre de cases intérieures.

		
cycle à l'origine	en vert les transformations (une 1 et trois 2)	en jaune la seconde serie transformation. On arrive a un rectangle

Lors de chaque transformation on créé un rectangle de dominos, donc on encadre un nombre impair de cases. À celles-ci on rajoute la ligne créée, de longueur impaire elle aussi. Au final on a soustrait au nombre de cases encadrées à l'origine un nombre pair (voir le lemme en annexe). Donc la parité des cases encadrées est invariante par les transformations considérées.
Finalement la présence d'un cycle n'est pas compatible avec le pavage.

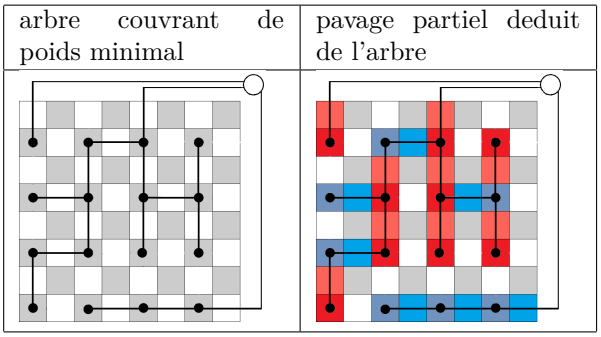
Reste à montrer que cet arbre est couvrant. Soit s un sommet de G_p. Le domino incluant ce sommet est le fils d'un autre sommet de G_p, s_1. Si ce sommet est dans T_p alors on doit ajouter s car on a traversé un seul bord. Sinon le domino incluant s_1 est fils d'un autre sommet de G_p et ce n'est pas s sinon on aurait un cycle. En réitérant cette méthode on forme alors un ensemble de dominos bien posés, et l'on ne peut arrêter ce processus que lorsqu'on atteint le sommet extérieur (on arrêtera avant si l'on tombe sur un sommet de T_p mais le raisonnement reste le même) et qu'on ne forme pas de cycle d'après le lemme. Mais alors la suite de sommets de ce graphe est à ajouter à T_p car on traverse exactement un bord de domino entre chaque sommet et donc s est dans T_p.
De la même manière on peut construire T_i un graphe couvrant de G_i. On notera que, mis à part pour les arêtes menant au sommet extérieur, T_p et T_i ne se chevauchent pas, car sinon on aurait un domino à la fois vertical et horizontal. On peut donc déduire d'un pavage par dominos un arbre couvrant de G_p.



Pavage du carré 8×8 et arbres couvrants déduits

Montrons maintenant que la donnée seule d'un arbre couvrant de G_p permet bien de construire un pavage par dominos.
Partant du sommet extérieur on pose un domino touchant le bord depuis les sommets voisins du sommet extérieur. Ensuite la règle est que pour chaque voisin non visité d'un sommet visité on pose un domino fils de l'orientation de l'arête et ayant pour extrémité le sommet non visité. Ainsi par récurrence on pose un pavage incomplet avec la motié des dominos du pavage. Le reste des dominos ne peut être mis que d'une seule manière. On les pose en construisant l'arbre T_i : partant du sommet extérieur impair on pose un domino fils vers un voisin si aucun domino déjà posé ne l'empêche. On

reconstruit alors bien un arbre couvrant de G_i. En effet il est couvrant car si on ne pouvait pas à partir d'un sommet quelconque de G_i arriver au sommet extérieur alors on aurait un cycle de dominos dans T_p ce qui est absurde et c'est bien un arbre car si l'on avait un cycle on aurait des sommets de T_p inaccessibles. Finalement on a recréé un unique pavage à partir d'un arbre couvrant de G_p.



Le sens arbre vers pavage de cette bijection est codé dans le programme `dessin` pour le pavage partiel à partir de l'arbre T_p et `arbre_dual_dessin` pour la création de l'arbre impair et le dessin du pavage induit. On trouvera ces programmes en annexe A. On va maintenant chercher à obtenir ces pavages via la création d'arbres couvrants aléatoires de la grille G_p.

3 Arbre couvrant de poids minimal

Pour obtenir de manière aléatoire un arbre couvrant on pense en premier lieu à attribuer un poids aléatoire à chaque arête puis à appliquer un algorithme de recherche d'un arbre couvrant de poids minimal comme l'algorithme de *Kruskal* ou celui de *Borůvka*. Notre choix se porte sur l'algorithme de *Prim-Janrnìc* car il crée à partir d'une racine donnée un arbre qui est toujours inclu dans l'arbre résultat. On peut alors, si l'on fait partir l'algorithme du sommet extérieur, dessiner le pavage au fur et à mesure que l'arbre s'agrandit ce qui permet un gain de temps dans les calculs. On va donner une idée de l'algorithme en pseudo-code.
Mais d'abord ce lemme qui assure l'existence d'un arbre couvrant de poids minimal :

Lemme 2 *Soit $G = (V, E)$ un graphe pondéré simple et connexe. Alors il existe un arbre couvrant de poids minimal pour la relation d'ordre sur les poids.*

démonstration : On construit des arbres couvrants algorithmiquement en supprimant à chaque étape une arête d'un circuit. Ainsi on conserve la connexité et on réduit de 1 le nombre de circuits. Quand on n'a plus de cicuit on a un arbre. Il n'y a qu'un nombre fini de manière d'enlever, donc un nombre fini d'arbres couvrants. Parmi ceux-ci on en choisit un dont la somme des poids des arêtes est minimal.

Remarque : on n'a pas l'unicité en général.

Data: graphe G=(V,E) ; racine ext
Result: arbre T
$F := \{ext\}$
$A := \emptyset$
while *cardF* $\neq n$*(*tant qu'on n'a pas tous les sommets*)* **do**
    choisir une arête dans E telle que
    $uv \in E, u \in F, v \notin F, w(uv) = \min_E\{w(ab), a \in F, b \in V \setminus F, ab \in E\}$;
    $F := F \bigcup \{v\}$;
    $A := A \bigcup \{uv\}$;
    Poser un domino de u vers v incluant v ;
end

Algorithm 1: Prim

On va montrer que l'algorithme termine et renvoie bien un arbre, qui plus est couvrant et de poids minimal. L'idée est d'inclure l'arbre créé à la k-ième étape dans un arbre couvrant de poids minimal, ainsi à la dernière étape l'arbre renvoyé est inclu et de même taille qu'un arbre couvrant de poids minimal, c'est donc un arbre couvrant de poids minimal.

Théorème 3 *L'algorithme de Prim appliqué à un graphe pondéré simple et connexe G termine et renvoie un arbre couvrant de poids minimal pour le graphe G*

démonstration :
-terminaison : comme il existe un arbre couvrant, on peut le construire par ajouts de sommet sans création de circuit. L'ordre ne nous intéresse pas.

-arbre couvrant de poids minimal : Soit T un arbre couvrant de poids minimal qui contient T_k, l'arbre construit par l'algorithme à la k-ième étape. On appelle uv l'arête ajoutée par l'algorithme à la prochaine étape, donc $u \in T_k, v \notin T_k$. Si $uv \in T$ on n'a rien à faire. Sinon uv induit un circuit dans T. On a un circuit $u, s_1, \ldots, s_k, v$ un circuit élémentaire dans $T \cup uv$. Les sommets s_1 à s_{d-1} et u sont dans $T_k \cap T$ et les sommets s_d à s_k et v sont dans $T \backslash T_k$. Mais alors l'arête $s_{d-1}s_d$ est de poids égal à celui de uv, au vu de l'algorithme et de la minimalité de T. Donc $T \backslash s_{d-1}s_d \cup uv$ est un arbre couvrant de poids minimal et contient T_{k+1}.

Cet algorithme est implémenté en `Scilab` par le programme `min_span_tree`. On n'a pas retenu l'idée de dessiner le pavage au fur et à mesure pour conserver la généralité de l'algorithme.
L'inconvénient de cette méthode est sa mauvaise complexité : on doit pondérer aléatoirement toutes les arêtes du graphe G_p puis appliquer l'algorithme de *Prim* qui, avec ses structures de données, tourne en $O(n^6)$ avec n le coté du carré. La recherche de minimum est coûteuse et on pourrait chercher à s'en passer par l'agorithme de Wilson [2].

4 Programmmes

Dans tous les programmes on a pris comme option de représenter les graphes par leurs matrices d'adjacence bien que ce ne soit pas la manière optimale pour des graphes plutôt creux. Cependant ce choix se justifie par le besoin d'accèder vite à un sommet particulier dans le programme de dessin et encore plus dans celui de construction de l'arbre dual. Il faudrait pour améliorer l'ensemble travailler à la création de l'arbre couvrant avec une autre structure de données (liste d'adjacence ou tas), puis de repasser à la forme matricielle pour le dessin. Un autre choix peu habituel est de représenter l'absence d'arête par un grand nombre. Ce choix est motivé par la pondération aléatoire du graphe.

On commence par le programme créant la matrice aléatoire. Ce programme construit une matrice représentant le graphe sans arête, puis met en place la sur-diagonale, sauf les sommets des bords.

```
//////////////
// Memoire //
//////////////

// Matrice grille aleatoire //
//*on relie les sommets de maniere aleatoire s'ils a une distance unite *//
//    o-o-o-o-     1 -2 -3 -4 -ext
//    | | | |      |  |  |  |
//    o-o-o-o-     5 -6 -7 -8 -ext
//    | | | |      |  |  |  |
//    o-o-o-o-     9 -11-11-12-ext
//    | | | |      |  |  |  |
//    o-o-o-o-     13-14-15-16-ext
//    | | | |      |  |  |  |

function randM = rand_mat (n,l,h) //taille, minimum, maximum

n2=n*n;

randM=11111*ones(n2+1,n2+1); //matrice aleatoire


EXT=11111*ones(n2+1,1); //sommet exterieur
for i=1:(n2)
    if modulo(i,n)==1 then
        EXT(i)=grand(1,1,'unf',l,h);
    end
    if (n2-i)<n then
        EXT(i)=grand(1,1,'unf',l,h);
    end
end
EXT(n2+1)=11111;

VOISL=11111*ones(n2,1);
VOISC=11111*ones(n2-n+1,1);


randM=randM-diag(VOISL,1)-diag(VOISL,-1)-diag(VOISC,n)-diag(VOISC,-n);

for i=1:(n2-1)
    if ~(modulo(i,n)==1) then
        VOISL(i)=grand(1,1,'unf',l,h);
    end
end

VOISC=grand(n2-n+1,1,'unf',l,h);
VOISC(n2-n+1)=11111;
```

```
randM=randM+diag(VOISL,1)+diag(VOISL,-1)+diag(VOISC,n)+diag(VOISC,-n);
randM(:,n2+1)=EXT;
randM(n2+1,:)=EXT';

endfunction
```

On programme ensuite l'algorithme de *Prim*, assez naïvement vu le choix de représentation des graphes.

```
//algorithme de Prim//
//codage loin d'etre optimal dans la recherche de l'arete minimale

//* les graphes seront vus via leurs matrices d'adjacence*//


function verticesweight = mini(graph,seen)  //arete a ajouter
                                      //(sommet non visite,sommet visite,poids)
    verticesweight=[1,1,1];
    n=length(seen);
    m=11111;
    k=1;
    h=1;
    for j = 1:n //on visite par colonne

        if seen(j) then    //pour tous les sommets deja dans l'arbre
            for i= 1:n     // pour tous les voisins de ces sommets
                if (graph(i,j))< m then
                    m=(graph(i,j));
                    verticesweight=[i,j,m];
                end
            end
        end


endfunction;

function tree=min_span_tree(graph,root)
    vmax=11111;
    n=sqrt(length(graph));
    graphe=graph;  //copie de travail du graphe
    tree=vmax*ones(n,n); //arbre resultat
    seen=zeros(1,n);
    seen(root)=1;
    graphe(root,:)=vmax*ones(1,n); //on ne pourra pas revenir au sommet racine
    for i = 1:(n-1)  //boucle principale

        vvw=zeros(1,3);
```

```
        vvw(1,:)=(mini(graphe,seen));

        tree(vvw(1),vvw(2))=vvw(3); //on ajoute l'arete
        tree(vvw(2),vvw(1))=vvw(3); //idem
        graphe(vvw(1),:)=vmax*ones(1,n); //mise a jour du graphe

        seen(vvw(1))=1;

    end

endfunction
```

Le programme de dessin de la partie paire du pavage est plutôt simple à concevoir à la main, les difficultés pratiques viennent de la gestion du dessin (la première partie `draw_dom`) et des cas particuliers sur les bords.
Le programme est récursif comme cela apparait dans l'algorithme : on examine un sommet puis on examine chacun de ses fils.

```
//Dessin//


//fonction de dessin des dominos
function i= draw_dom(l,n,o,e) //longueur origine extremite
    a=gca()
    a.isoview='on'
    L=2*l;
    if (abs(o-e))==1 then //domino horizontal
        if (o-e)==1 then //domino gauche
                xset("color",6)
                xfrect(modulo(e-1,n)*2*l+l,-(floor((e-1)/n))*2*l,l,l)
                xset("color",5)
                xfrect(modulo(e-1,n)*2*l,-(floor((e-1)/n))*2*l,l,l)
                xset("color",1)
                xrect(modulo(e-1,n)*2*l,-(floor((e-1)/n))*2*l,L,l)

        else              //domino droit
                xset("color",6)
                xfrect(modulo(e-1,n)*2*l-l,-(floor((e-1)/n))*2*l,l,l)
                xset("color",5)
                xfrect(modulo(e-1,n)*2*l,-(floor((e-1)/n))*2*l,l,l)
                xset("color",1)
                xrect(modulo(e-1,n)*2*l-l,-(floor((e-1)/n))*2*l,L,l)

        end
    else             //domino vertical
        if (o-e)>1 then //domino haut
        xset("color",4)
        xfrect(modulo(e-1,n)*2*l,-floor((e-1)/n)*2*l-l,l,l)
        xset("color",2)
```

```
        xfrect(modulo(e-1,n)*2*l,-floor((e-1)/n)*2*l,l,l)
        xset("color",1)
        xrect(modulo(e-1,n)*2*l,-floor((e-1)/n)*2*l,l,L)
        else              //domino bas
        xset("color",4)
        xfrect(modulo(e-1,n)*2*l,-floor((e-1)/n)*2*l+l,l,l)
        xset("color",2)
        xfrect(modulo(e-1,n)*2*l,-floor((e-1)/n)*2*l,l,l)
        xset("color",1)
        xrect(modulo(e-1,n)*2*l,-floor((e-1)/n)*2*l+l,l,L)
        end
    end
    //sleep(1111);
    i=1;
endfunction


function e = draw_tree_aux(l,tree,N,n,i,o) // longueur, arbre, sommet examine,
sommet d'origine
// n>3 pour eviter les cas particuliers
    a=gca()
    a.isoview='on'


  if(~(modulo(i,n)==1)) then

    if tree(i,i-1)<5111 & ~((i-1)==o) then

      draw_dom(l,n,i,i-1);
      draw_tree_aux(l,tree,N,n,i-1,i);

    end

  end

  if(~(modulo(i,n)==-1)) then
    if tree(i,i+1)<5111 & ~((i+1)==o) then
      draw_dom(l,n,i,i+1);
      draw_tree_aux(l,tree,N,n,i+1,i);

    end

  end

  if(i>n) then
    if tree(i,i-n)<5111 & ~((i-n)==o) then
      draw_dom(l,n,i,i-n);
      draw_tree_aux(l,tree,N,n,i-n,i);
```

```
    end

  end

  if(i<N-n) then
    if tree(i,i+n)<5111 & ~((i+n)==o) then
      draw_dom(l,n,i,i+n);
      draw_tree_aux(l,tree,N,n,i+n,i);

    end

  end

  e=1;

endfunction



function b = draw_tree(l,tree)


    plot2d(1,1,-1,"111","␣",[-5,-5,5,5])
    a=gca();
    a.isoview='on';
    N=sqrt(length(tree)); //nombre de sommets
    n=sqrt(N-1);          //cote de la grille

    for i=1:(2*n) //quadrillage

        xrect(i*l,1,l/51,2*l*n) //lignes verticales
        xrect(1,-i*l,2*l*n,l/51)//lignes horizontales
    end

    for k = 1:(n-2)
        if tree(n+k*n,N)< 5111 then //bord droit
            draw_dom(l,n,n+k*n+1,n+k*n);                                    //**||poser
 un domino droit
            draw_tree_aux(l,tree,N,n,n+k*n,N);
        end
        if tree(N-n+k,N)<5111 then
            draw_dom(l,n,N,N-n+k);                                  //**||poser un
domino haut
            draw_tree_aux(l,tree,N,n,N-n+k,N);
        end

    end
```

```
    if tree(N-1,N)<5111 then
  //on tire a pile ou face pour savoir comment on place le domino de coin
        b=grand(1,1,'bin',1,1.5);
        if b then
            draw_dom(l,n,N,N-1);                                    //**||poser un
domino droit

        else
            draw_dom(l,n,N+n,N-1);                                   //**||poser un
domino haut
                                                                    //on "triche" pour
donner un domino vertical
        end
        draw_tree_aux(l,tree,N,n,N-1,N);
    else
         b=-1;
    end


endfunction
```

Enfin le dernier programme présenté construit l'arbre impair tout en le dessinant. Là encore les difficultés sont dans le dessin lui-même et la gestion des bords.

```
//fonction de dessin de l'arbre impair

function i= draw_dom2(l,n,o,e) //longueur origine extremite
    a=gca()
    a.isoview='on'
    L=2*l;
    if (abs(o-e))==1 then //domino horizontal
        if (o-e)==1 then //domino gauche
                xset("color",6)
                xfrect(modulo(e-1,n)*2*l+2*l,-(floor((e-1)/n))*2*l-l,l,l)
                xset("color",5)
                xfrect(modulo(e-1,n)*2*l+l,-(floor((e-1)/n))*2*l-l,l,l)
                xset("color",1)
                xrect(modulo(e-1,n)*2*l+l,-(floor((e-1)/n))*2*l-l,L,l)

        else              //domino droit
                xset("color",6)
                xfrect(modulo(e-1,n)*2*l,-(floor((e-1)/n))*2*l-l,l,l)
                xset("color",5)
                xfrect(modulo(e-1,n)*2*l+l,-(floor((e-1)/n))*2*l-l,l,l)
                xset("color",1)
                xrect(modulo(e-1,n)*2*l,-(floor((e-1)/n))*2*l-l,L,l)

        end
```

```
    else             //domino vertical
        if (o-e)>1 then //domino haut
        xset("color",4)
        xfrect(modulo(e-1,n)*2*l+l,-floor((e-1)/n)*2*l-2*l,l,l)
        xset("color",2)
        xfrect(modulo(e-1,n)*2*l+l,-floor((e-1)/n)*2*l-l,l,l)
        xset("color",1)
        xrect(modulo(e-1,n)*2*l+l,-floor((e-1)/n)*2*l-l,l,L)
        else              //domino bas
        xset("color",4)
        xfrect(modulo(e-1,n)*2*l+l,-floor((e-1)/n)*2*l,l,l)
        xset("color",2)
        xfrect(modulo(e-1,n)*2*l+l,-floor((e-1)/n)*2*l-l,l,l)
        xset("color",1)
        xrect(modulo(e-1,n)*2*l+l,-floor((e-1)/n)*2*l,l,L)
        end
    end
    //sleep(1111);
    i=1;
endfunction

//construction de l'arbre dual tree2 selon Temperley
//    | | | |      |  |  |  |
//   -o-o-o-o-     1 -2 -3 -4 -ext
//    | | | |      |  |  |  |
//   -o-o-o-o-     5 -6 -7 -8 -ext
//    | | | |      |  |  |  |
//   -o-o-o-o-     9 -11-11-12-ext
//    | | | |      |  |  |  |
//   -o-o-o-o-     13-14-15-16-ext
//
//dessin en meme temps

function t= dualtree_aux2 (tree,tree2,b,N,n,o,e)
    t=tree2;
    //on traite les cas aux bords
    if floor((e-1)/n)==1 then //premiere ligne
        if modulo(e,n)==1 then //derniere colone

            if (tree(e+n,N)>5111 & ~((e+n)==o)) then //bas
                t(e,e+n)=1;
                t(e+n,e)=1;
                draw_dom2(l,n,e,e+n);
                t=dualtree_aux2(tree,t,b,N,n,e,e+n);
            end
            if (tree(e,e+n)>5111 & ~((e-1)==o)) then
                t(e,e-1)=1;
                t(e-1,e)=1;
                draw_dom2(l,n,e,e-1);
```

```
            t=dualtree_aux2(tree,t,b,N,n,e,e-1);
        end
    else
        if modulo(e,n)==1 then //premiere colone
         if (tree(e+n,e+n+1)>5111 & ~((e+n)==o)) then //haut
               t(e,e+n)=1;
               t(e+n,e)=1;
               draw_dom2(l,n,e,e+n);
               //on ne peut pas aller plus loin
         end
         if (tree(e+1,e+1+n)>5111 & ~((e+1)==o)) then //droite
               t(e,e+1)=1;
               t(e+1,e)=1;
               draw_dom2(l,n,e,e+1);
               t=dualtree_aux2(tree,t,b,N,n,e,e+1);
         end


        else //cas general, premiere ligne

            if (tree(e+1,e+1+n)>5111 & ~((e+1)==o)) then //droite
                t(e,e+1)=1;
                t(e+1,e)=1;
                draw_dom2(l,n,e,e+1);
                t=dualtree_aux2(tree,t,b,N,n,e,e+1);
            end
            if (tree(e+n,e+n+1)>5111 & ~((e+n)==o)) then //bas
                t(e,e+n)=1;
                t(e+n,e)=1;
                draw_dom2(l,n,e,e+n);
                t=dualtree_aux2(tree,t,b,N,n,e,e+n);
            end
            if (tree(e,e+n)>5111 & ~((e-1)==o)) then
                t(e,e-1)=1;
                t(e-1,e)=1;
                draw_dom2(l,n,e,e-1);
                t=dualtree_aux2(tree,t,b,N,n,e,e-1);
            end
        end

     end
else
    if floor((e-1)/n)== (n-1) then //derniere ligne
        if modulo(e,n)==1 then //premiere colonne
            if (tree(e,e+1)>5111 & ~((e-n)==o)) then //haut
                t(e,e-n)=1;
                t(e-n,e)=1;
                draw_dom2(l,n,e,e-n);
                t=dualtree_aux2(tree,t,b,N,n,e,e-n);
```

```
    end
    if (tree(e+1,N)>5111 & ~((e+1)==o)) then //droite
        t(e,e+1)=1;
        t(e+1,e)=1;
        draw_dom2(l,n,e,e+1);
        t=dualtree_aux2(tree,t,b,N,n,e,e+1);
    end
else
    if  e==(n*n-1) then //avant derniere colonne, depend de b
       if (tree(e,e+1)>5111 & ~((e-n)==o)) then //haut
                t(e,e-n)=1;
                t(e-n,e)=1;
                draw_dom2(l,n,e,e-n);
                t=dualtree_aux2(tree,t,b,N,n,e,e-n);
            end
            if (b==-1|b==1) & ~((e+1)==o) then //droit
                t(e,e+1)=1;
                t(e+1,e)=1;
                draw_dom2(l,n,e,e+1);
                t=dualtree_aux2(tree,t,b,N,n,e,e+1);
            end
            if (tree(e,N)>5111 & ~((e-1)==o)) then //gauche
                t(e,e-1)=1;
                t(e-1,e)=1;
                draw_dom2(l,n,e,e-1);
                t=dualtree_aux2(tree,t,b,N,n,e,e-1);
            end
    else
        if modulo(e,n)==1 then //derniere colonne; ! depend de b
            if o==(e-1)  then
                if b==-1 | b==1 then
                    t(e,e-n)=1;
                    t(e-n,e)=1;
                    draw_dom2(l,n,e,e-n);
                    t=dualtree_aux2(tree,t,b,N,n,e,e-n);
                end
            else
                if b==-1 | b==1 then
                    t(e,e-1)=1;
                    t(e-1,e)=1;
                    draw_dom2(l,n,e,e-1);
                    t=dualtree_aux2(tree,t,b,N,n,e,e-1);
                end
            end

        else // cas general, derniere ligne
            if (tree(e,e+1)>5111 & ~((e-n)==o)) then //haut
                t(e,e-n)=1;
                t(e-n,e)=1;
```

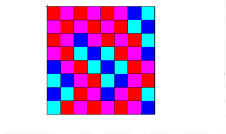
```
                draw_dom2(l,n,e,e-n);
                t=dualtree_aux2(tree,t,b,N,n,e,e-n);
            end
            if (tree(e+1,N)>5111 & ~((e+1)==o)) then //droit
                t(e,e+1)=1;
                t(e+1,e)=1;
                draw_dom2(l,n,e,e+1);
                t=dualtree_aux2(tree,t,b,N,n,e,e+1);
            end
            if (tree(e,N)>5111 & ~((e-1)==o)) then //gauche
                t(e,e-1)=1;
                t(e-1,e)=1;
                draw_dom2(l,n,e,e-1);
                t=dualtree_aux2(tree,t,b,N,n,e,e-1);
            end
        end
    end

end
else
    if modulo(e,n)==1 then //derniere colone,
        if e==(n*(n-1)) then // depend de b
            if (tree(e,N)>5111 & ~((e-n)==o)) then //haut
                t(e,e-n)=1;
                t(e-n,e)=1;
                draw_dom2(l,n,e,e-n);
                t=dualtree_aux2(tree,t,b,N,n,e,e-n);
            end
            if (tree(e,e+n)>5111 & ~((e-1)==o)) then //gauche
                t(e,e-1)=1;
                t(e-1,e)=1;
                draw_dom2(l,n,e,e-1);
                t=dualtree_aux2(tree,t,b,N,n,e,e-1);
            end
            if (b==-1|b==1) & ~((e+n)==o) then //bas
                t(e,e+n)=1;
                t(e+n,e)=1;
                draw_dom2(l,n,e,e+n);
                t=dualtree_aux2(tree,t,b,N,n,e,e+n);
            end
        else //cas general
            if (tree(e,N)>5111 & ~((e-n)==o)) then //haut
                t(e,e-n)=1;
                t(e-n,e)=1;
                draw_dom2(l,n,e,e-n);
                t=dualtree_aux2(tree,t,b,N,n,e,e-n);
            end
            if (tree(e,e+n)>5111 & ~((e-1)==o)) then //gauche
                t(e,e-1)=1;
```

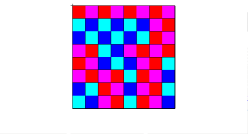
```
            t(e-1,e)=1;
            draw_dom2(l,n,e,e-1);
            t=dualtree_aux2(tree,t,b,N,n,e,e-1);
        end
        if (tree(e+n,N)>5111 & ~((e+n)==o)) then //bas
            t(e,e+n)=1;
            t(e+n,e)=1;
            draw_dom2(l,n,e,e+n);
            t=dualtree_aux2(tree,t,b,N,n,e,e+n);
        end
    end

else
    if modulo(e,n)==1 then //premiere colone, cas general
        if (tree(e,e+1)>5111 & ~((e-n)==o)) then //haut
            t(e,e-n)=1;
            t(e-n,e)=1;
            draw_dom2(l,n,e,e-n);
            t=dualtree_aux2(tree,t,b,N,n,e,e-n);
        end
        if (tree(e+1,e+n+1)>5111 & ~((e+1)==o)) then //droite
            t(e,e+1)=1;
            t(e+1,e)=1;
            draw_dom2(l,n,e,e+1);
            t=dualtree_aux2(tree,t,b,N,n,e,e+1);
        end
        if (tree(e+n,e+n+1)>5111 & ~((e+n)==o)) then //bas
            t(e,e+n)=1;
            t(e+n,e)=1;
            draw_dom2(l,n,e,e+n);
            t=dualtree_aux2(tree,t,b,N,n,e,e+n);
        end
        //pas de gauche, traite par dualtree
    else //cas general, sans aucun bord
            if (tree(e,e+1)>5111 & ~((e-n)==o)) then //haut
                t(e,e-n)=1;
                t(e-n,e)=1;
                draw_dom2(l,n,e,e-n);
                t=dualtree_aux2(tree,t,b,N,n,e,e-n);
            end
            if (tree(e+1,e+1+n)>5111 & ~((e+1)==o)) then //droite
                t(e,e+1)=1;
                t(e+1,e)=1;
                draw_dom2(l,n,e,e+1);
                t=dualtree_aux2(tree,t,b,N,n,e,e+1);
            end
            if (tree(e+n,e+n+1)>5111 & ~((e+n)==o)) then //bas
                t(e,e+n)=1;
                t(e+n,e)=1;
```

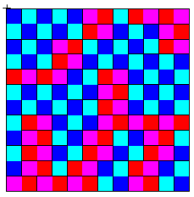
```
                draw_dom2(l,n,e,e+n);
                t=dualtree_aux2(tree,t,b,N,n,e,e+n);
            end
            if (tree(e,e+n)>5111 & ~((e-1)==o)) then
                t(e,e-1)=1;
                t(e-1,e)=1;
                draw_dom2(l,n,e,e-1);
                t=dualtree_aux2(tree,t,b,N,n,e,e-1);
            end
        end

    end

end

endfunction



function tree2=dualtree2(l,tree,b) //le b correspond au choix d'orientation pour le
domino bas gauche de tree
    N=sqrt(length(tree)); //nombre de sommets
    n=sqrt(N-1);          //cote de la grille

    tree2=ones(N,N)*11111; //arbre vide

    if tree(1,2)>5111 then   //premier domino, haut gauche
    tree2(1,N)=+1;            //on differencie le signe pour l'orientation du domino
    tree2(N,1)=+1;
    draw_dom2(l,n,-1,1);
    tree2=dualtree_aux2(tree,tree2,b,N,n,N,1);

    elseif tree(1,n+1)>5111 then
    tree2(1,N)=-1
    tree2(N,1)=-1
    draw_dom2(l,n,1,1);
    tree2=dualtree_aux2(tree,tree2,b,N,n,N,1);

    else
    {};
    end

    for i=2:(n-1)
        if tree(i,i+1)>5111 then //premiere ligne

        tree2(i,N)=1;
        tree2(N,i)=1;
```

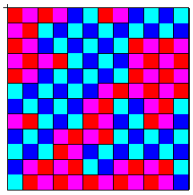
```
        draw_dom2(l,n,-1,i);
        tree2=dualtree_aux2(tree,tree2,b,N,n,N,i);

        end
        if tree(1+n*(i-1),n*i+1)>5111 then //premiere colonne

        tree2(1+n*(i-1),N)=1;
        tree2(N,1+n*(i-1))=1;
        draw_dom2(l,n,n*(i-1),1+n*(i-1));
        tree2=dualtree_aux2(tree,tree2,b,N,n,N,1+n*(i-1));

        end

    end

    if tree(n,N)>5111 then

        tree2(n,N)=1;
        tree2(N,n)=1;
        draw_dom2(l,n,-1,n);
        tree2=dualtree_aux2(tree,tree2,b,N,n,N,n);

    end

    if tree(N-n,N)>5111 then

        tree2(N-n,N)=1;
        tree2(N,N-n)=1;
        draw_dom2(l,n,N-n-1,N-n);
        tree2=dualtree_aux2(tree,tree2,b,N,n,N,N-n);

    end

endfunction
```

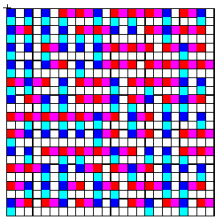
Au final le coût du programme vient largement de la construction de l'arbre couvrant. Le dessin en lui même prend un parcours de chaque sommet, de même pour le programme de construction et de dessin de l'arbre impair. Cependant les cas particuliers aux bords et le dessin dans `Scilab` sont assez lourds visuellement ce qui rend le programme assez long et inquiétant si l'on ne connaît pas l'approche théorique.
Voici quelques dessins obtenus par ces programmes :



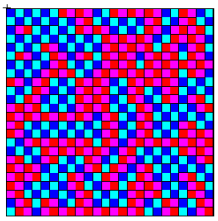




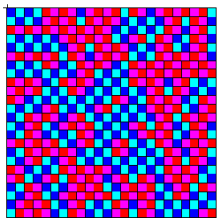


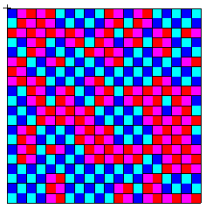


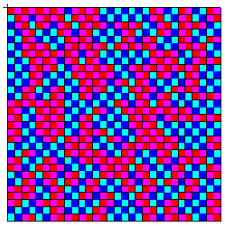
juste le pavage de l'arbre pair

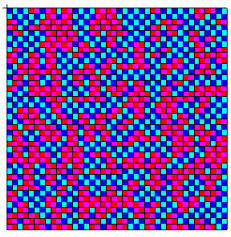


pavage complet









A Annexes

A.1 Lemmes techniques

Lemme 4 *Dans un cycle de dominos codé par les lettres s, r, l la valeur absolue de la différence du nombre de r et du nombre de l vaut 4*

démonstration : On va noter n_r le nombre de r et n_l le nombre de l, ainsi que $n = n_r + n_l$. Le cycle de dominos peut être vu comme un polygone non croisé où chaque r ou l correspond à un sommet. On considère qu'un r correspond à un angle de $\pi/2$ et un l à $3\pi/2$. Cela nous définit un intérieur et un extérieur§. Si l'intérieur est fini la somme des angles vaut $\pi(n-2)$. Sinon elle est égale à $\pi(n+2)$.

On résout le système :

$$\begin{cases} n_r + n_l = n \\ \frac{\pi}{2} n_r + 3\frac{\pi}{2} n_l = \pi(n \pm 2) \end{cases} \iff \begin{cases} n_r + n_l = n \\ n_r - n_l = \pm 4 \end{cases}$$

Ce qui donne bien le résultat attendu : $|n_r - n_l| = 4$.

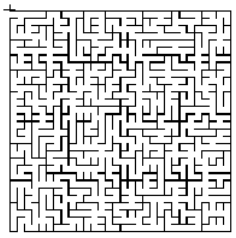
Lemme 5 *Dans un cycle de dominos rectangle, c'est à dire de la forme $rs^i rs^j rs^i rs^j$ (où $ls^i ls^j ls^i ls^j$),avec i et j entiers naturels, les côtés sont formés d'un nombre impair de cases et le rectangle encadre un nombre lui aussi impair de cases.*

démonstration : Chaque côté est constitué d'une ligne de s, dominos complets constitués de 2 cases, d'un domino d'après virage complet et de la dernière case du domino de virage. Donc le coté a un nombre impair de cases.

Le nombre de cases intérieures est donnée par l'aire intérieure, c'est à dire l'aire totale moins l'aire des dominos du cycle. L'aire des dominos du cycle est paire car chaque domino est constitué de deux cases et l'aire totale est impaire car chaque coté à un nombre impair de cases.

un rectangle de dominos, $rsrrsr$

A.2 Labyrinthe



un labyrinthe de coté 30 tracé par le programme

Une application plus simple des arbres couvrants de poids minimal sur des graphes grille est la création de labyrinthes. Là encore on peut trouver des applications à la physique statistique. De plus on a ainsi une nouvelle bijection entre les pavages et les labyrinthes, via les arbres.

```
//arbre adapte au labyrinthe//
mode(1);

// Matrice grille aleatoire //
//*on relie les sommets de maniere aleatoire s'ils sont a une distance unite *//
//    o-o-o-o     1 -2 -3 -4
//    | | | |     |  |  |  |
//    o-o-o-o     5 -6 -7 -8
//    | | | |     |  |  |  |
//    o-o-o-o     9 -11-11-12
//    | | | |     |  |  |  |
//    o-o-o-o     13-14-15-16
//
//


function randM = rand_mat_maze (n,l,h) //taille, minimum, maximum

n2=n*n;

randM=11111*ones(n2,n2); //matrice aleatoire


VOISL=11111*ones(n2-1,1);
VOISC=11111*ones(n2-n,1);


randM=randM-diag(VOISL,1)-diag(VOISL,-1)-diag(VOISC,n)-diag(VOISC,-n);

for i=1:(n2-1)
    if ~(modulo(i,n)==1) then
        VOISL(i)=grand(1,1,'unf',l,h);
    end
end

VOISC=grand(n2-n,1,'unf',l,h);
randM=randM+diag(VOISL,1)+diag(VOISL,-1)+diag(VOISC,n)+diag(VOISC,-n);


endfunction
```

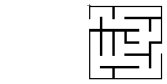
Pour le dessin du labyrinthe on trace un rectangle en noir puis on creuse le chemin en suivant l'arbre.

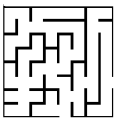
```
//dessin d'un labyrinthe//
mode(1);

// 1=gauche, 2=bas, 3=droite, 4=haut
```

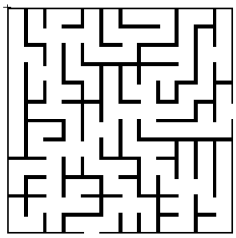
```
// 1-2-3
// | | |
// 4-5-6
// | | |
// 7-8-9
// | | |


function i =dig(l,n,o,e)
    a=gca();

    xset("color",8)
    L=2*l;
    p=l/11;

    if (abs(o-e))==1 then // horizontal
        if (o-e)==1 then // gauche

            xfrect(modulo(e-1,n)*l+p,-(floor((e-1)/n))*l-p,L-2*p,l-2*p)


        else              //domino droit

            xfrect(modulo(e-1,n)*l-l+p,-(floor((e-1)/n))*l-p,L-2*p,l-2*p)

        end
    else             //domino vertical
        if (o-e)>1 then //domino haut

            xfrect(modulo(e-1,n)*l+p,-floor((e-1)/n)*l-p,l-2*p,L-2*p)
        else             //domino bas

            xfrect(modulo(e-1,n)*l+p,-floor((e-1)/n)*l+l-p,l-2*p,L-2*p)
        end
    end
    i=1
endfunction


function i= maze_dig(l,n,tree,o,e)
    a=gca();
    if ~(modulo(e,n)==1) then
        if tree(e,e-1)<5111 & ~((e-1)==o) then

          dig(l,n,e,e-1);
          maze_dig(l,n,tree,e,e-1);

        end
```

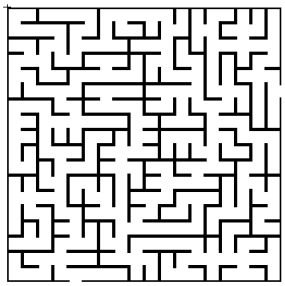
```
    end

    if ~(modulo(e,n)==1) then
        if  tree(e,e+1)<5111 & ~((e+1)==o) then

          dig(l,n,e,e+1);
          maze_dig(l,n,tree,e,e+1);

        end
    end

    if ~(floor((e-1)/n)>=(n-1))   then

        if tree(e,e+n)<5111 & ~((e+n)==o) then

          dig(l,n,e,e+n);
          maze_dig(l,n,tree,e,e+n);


        end
    end

    if ~(floor((e-1)/n)==1) then
        if tree(e,e-n)<5111 & ~((e-n)==o) then

          dig(l,n,e,e-n);
          maze_dig(l,n,tree,e,e-n);


        end
    end


    i=1
endfunction




function i=maze(l,tree,e,s) //longueur dessin, arbre, entree [cote,numero],sortie [
cote numero]

    plot2d(1,1,-1,"111","␣",[-5,-5,5,5])
    a=gca();
    a.isoview='on';
    N=sqrt(length(tree)); //nombre de sommets
    n=sqrt(N);          //cote de la grille
```

```
//    xset("color",2)
//    xfrect(-11,11,15,15); //fond

    xset("color",1)       // labyrinthe non creuse
    xfrect(1,1,n*l,l*n);

    xset("color",8)

    if e(1)==1 then //gauche
        dig(l,n,(e(2)-1)*n,(e(2)-1)*n+1);
        maze_dig(l,n,tree,(e(2)-1)*n,(e(2)-1)*n+1);
    end
    if e(1)==3 then //droite
        dig(l,n,(e(2))*n+1,(e(2))*n);
        maze_dig(l,n,tree,(e(2))*n+1,(e(2))*n);
    end
    if e(1)==2 then //bas
        dig(l,n,N+e(2),N-n+e(2));
        maze_dig(l,n,tree,N+e(2),N-n+e(2));
    end
    if e(1)==4 then  //haut
        dig(l,e(2)-n,e(2));
        maze_dig(l,n,tree,e(2)-n,e(2));
    end


    if s(1)==1 then //gauche
        dig(l,n,(s(2)-1)*n,(s(2)-1)*n+1);

    end
    if s(1)==3 then //droite
        dig(l,n,(s(2))*n+1,(s(2))*n);

    end
    if s(1)==2 then //bas
        dig(l,n,N+s(2),N-n+s(2));

    end
    if s(1)==4 then  //haut
        dig(l,n,s(2)-n,s(2));

    end

    i=1;
endfunction
```









Références

[1] H. Temperley, *combinatorics proc. 1974.*

[2] D. B. Wilson, *Generating random spanning trees more quickly than the cover time*, Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996), 1996.

[3] T. De La Rue, E. Janvresse, *Pavage Aléatoire par Touillage de Domino*. image CNRS. 2009. `http://images.math.cnrs.fr/Pavages-aleatoires-par-touillage.html`

[4] W. P. Thurston, Conway's tiling groups Amer. Math. Monthly 97, 1990