UNIVERSITÉ PARIS 7 - DENIS DIDEROT
UFR D'INFORMATIQUE

RAPPORT DE STAGE

de 3^e année de l'IUP Génie Mathématiques et Informatique
soutenu par

Sylvain Sené

le 22 septembre 2004

Le modèle des piles de sable sur une grille bi-dimensionnelle

Responsable de stage : Roberto MANTACI
Tuteur de l'UFR : Yan JURSKI

Année universitaire 2003-2004

Remerciements

Il est indéniable que l'intégralité de ce stage n'aurait pu se dérouler dans de si bonnes conditions sans l'aide et le soutien d'un certain nombre de personnes travaillant toutes au LIAFA. En effet, de nombreux chercheurs ont su être présents pour que le projet puisse avancer de la meilleure façon possible.

Tout d'abord, je tiens particulièrement à remercier Roberto Mantaci, mon responsable de stage, qui a su me mettre à l'aise fort rapidement. Sa connaissance des différents modèles de tas de sable et les explications qu'il m'en a faites m'ont permis de rapidement m'accoutumer aux problèmes posés par le projet. Par ailleurs, il a su être à mon écoute aux moments où j'en avais besoin afin de m'aider à prendre les meilleures décisions.

Je souhaite aussi remercier Ha Duong Phan et Dominique Rossin qui ont toujours su répondre aimablement à chacune de mes questions relatives au projet qui m'a été confié, particulièrement quand M. Mantaci était absent.

Enfin, je tiens à ne pas oublier Inès Klimann et Christophe Prieur qui ont su me donner de bons conseils au sujet de la réalisation des logiciels, ainsi que tous ceux avec qui j'ai partagé des repas, des cafés et des discussions. Ces petits moments agréables et enrichissants sont pour beaucoup dans le plaisir que j'ai pris à travailler.

Table des matières

Introduction

Comme chaque année, l'ensemble des étudiants de l'IUP GMI s'est retrouvé confronté à la recherche d'un stage. En première et deuxième années, le stage représente, bien entendu, une étape importante puisqu'il peut déterminer le passage ou non au niveau supérieur. Les choses sont identiques pour la troisième année. Il a même un rôle encore plus important et son choix est réellement déterminant. Tout d'abord, quatre mois représentent une durée intéressante pour mener à bien un projet. Ensuite, il est évident que c'est au cours de la dernière année que nous devenons de "vrais" informaticiens, ayant acquis une certaine indépendance et expérience. De plus, pour l'étudiant désirant intégrer la vie active, ces quatre mois peuvent lui offrir des opportunités d'embauche alors que pour celui dont la volonté est de poursuivre ses études, il est préférable de travailler dans le domaine vers lequel il souhaite s'orienter dans le futur.

Depuis déjà quelques années, j'ai une idée générale de mes espérances professionnelles. Le stage de fin de DUT que j'ai effectué à l'Institut de la Communication Parlée (ICP) de Grenoble avant d'intégrer l'IUP m'a permis de découvrir le monde de la recherche et depuis lors, j'avoue connaître une réelle attirance pour le métier d'enseignant-chercheur. Ce stage, caractérisé par un projet d'informatique appliquée à la linguistique, ne m'a néanmoins pas donné l'occasion de comprendre en quoi consistait le métier d'enseignant-chercheur en informatique.

Ayant eu la chance de réaliser l'an dernier des *interviews* auprès de quatre enseignants de l'UFR d'informatique, j'ai pu obtenir certains renseignements comme leurs thèmes et intérêts de recherche. À ce moment précis, j'avais déjà en tête de réaliser mon stage de l'année suivante au sein de l'un des deux laboratoires de recherche en informatique de l'université. Par conséquent, vers le début du deuxième semestre de cette année, j'ai commencé à me renseigner auprès des professeurs que je connaissais déjà et dont je savais que le domaine de recherche m'intéresserait. M. Roberto Mantaci m'a alors proposé un sujet orienté vers la recherche qui correspondait entièrement à mes attentes car il était à la fois théorique et pratique, tout en ayant de réelles applications visibles dans le monde qui nous entoure.

Ce rapport présente les étapes et les résultats des quatre mois de travail (du 3 mai au 1^{er} septembre 2004) passés au sein de l'équipe d'algorithmique et combinatoire du Laboratoire d'Informatique et Algorithmique : Fondements et Applications (LIAFA) de Paris sur le thème des *SPM* (*Sand Pile Models*), systèmes dynamiques discrets simples utilisés en physique pour la représentation d'objets granulaires.

Ainsi, à la suite d'une brève présentation du contexte "logistique" sera introduit le problème général inhérent aux piles de sables. Ensuite, le deuxième chapitre sera consacré au cas particulier de la "trois-dimensions". Enfin, nous nous attacherons aux aspects plus pratiques de ce stage avant de dresser un bilan sur ces quatre mois.

1 Contexte et préliminaires

Le **modèle de pile de sable** (*Sand Pile Model*, *SPM*) est un moyen de représenter les systèmes granulaires. Il s'agit d'un système dynamique discret servant à modéliser les objets granulaires dans le domaine de la physique. Il peut être largement mis en relation, au niveau mathématique, avec les partitions d'entiers ainsi que de nombreux autres domaines combinatoires tels que les pavages et les systèmes de réécriture.

Dans ce chapitre, après avoir presenté brièvement l'organisme d'accueil de ce stage, nous expliquerons ce qui a amené les chercheurs à s'intéresser à ce problème pour ensuite nous attacher plus particulièrement aux recherches réalisées sur le cas bi-dimensionnel avant mon arrivée.

1.1 Le LIAFA

Le LIAFA est l'un des deux laboratoires (unité mixte de recherche) d'informatique en co-tutelle avec l'université Denis Diderot-Paris 7, et le CNRS (Centre National de la Recherche Scientifique). Il se compose de 3 thèmes de recherche et regroupe actuellement plus d'une trentaine d'enseignants-chercheurs et de chercheurs permanents ainsi qu'une vingtaine de doctorants et post-docs.

À l'heure actuelle, ce laboratoire de recherche est organisé scientifiquement autour de ses 3 équipes de recherche, à savoir

- l'équipe **Algorithmique et Combinatoire**,
- l'équipe **Automates et Applications**,
- ainsi que l'équipe **Modélisation et Vérification**

et d'un axe transversal **Systèmes à évènements discrets** qui regroupe des chercheurs issus de toutes les équipes du laboratoire.

Ces thématiques relèvent bien entendu toutes de l'algorithmique qui forme l'axe central autour duquel s'est construit le LIAFA. Elles correspondent de fait à autant de filières de l'ancien DEA d'Algorithmique, aujourd'hui intégré au Master Parisien de Recherche en Informatique (en co-habilitation avec l'université Denis Diderot, l'École Normale Supérieure de Paris, l'École Normale Supérieure de Cachan et l'École Polytechnique) dans lequel les membres du LIAFA sont plus que fortement impliqués.

Par ailleurs, l'activité scientifique du LIAFA est de tout premier plan dans de nombreux domaines fondamentaux de l'informatique, comme l'algorithmique des graphes, la combinatoire des permutations, les modèles du parallélisme, la théorie des automates ou encore la vérification, auxquels les chercheurs du laboratoire ont souvent contribuer de façon majeure.

Le LIAFA entretient également des rapports extrêmement étroits de collaboration avec les organismes français et étrangers apparaissant dans la liste ci-dessous :

- en France : les laboratoires de l'ENST (École Nationale Supérieure des Télécommunications), l'INRIA (Institut National de Recherche en Informatique et en Automatique) de Lorraine (Projet RESEDAS), l'INRIA de Rocquencourt (Projets Algo et Metalo), l'INRIA de Sophia-Antipolis (Projet SLOOP), le VERIMAG (laboratoire de VERification de l'institut d'Informatique et de Mathématiques Appliquées de Grenoble) (Projet DYNAMO)...
- en Europe : les universités d'Athènes, Berlin, Braga (Portugal), Budapest, Dresden, Liège, Lisbonne, Louvain la Neuve (Belgique), Mons-Hainaut (Belgique), Milan, Moscou, Porto, Prague, Stuttgart, Szeged (Hongrie), Turku (Finlande), York...
- en Amérique : les universités de Boston, Berkeley, Buenos Aires, Michigan, Montréal (UQAM), Santiago du Chili, Santa Fe, Sao Paulo, Stanford...
- en Afrique et au Moyen-Orient : les universités d'Antananarivo (Madagascar), Alger, Bar-Ilan (Israel)...
- en Asie : les universités de Chennai (Inde), Niigata (Japon)...

Le LIAFA a également engagé des collaborations suivies avec plusieurs partenaires industriels tels que Bouygues Telecom et Nortel Networks. Il est pareillement en train de s'engager dans de nouveaux partenariats industriels impliquant des sociétés à l'exemple d'Ericsson, Matra Systèmes & Information ou encore Siemens. La vie scientifique du LIAFA s'affirme enfin par la participation des membres du laboratoire à des formations de troisième cycle (MPRI, DESS Logiciels Fondamentaux de Paris 7) ainsi que par les groupes de travail hebdomadaires organisés par chaque équipe du laboratoire.

1.2 Les motivations du problème

En 1987, MM. Bak, Tang et Wiesenfeld [BTW87] ont introduit le concept de l'**auto-organisation critique** (*Self-Organization Criticality*, *SOC*). Ils remarquent que certains systèmes placés dans un état stable, nommé **état critique**, puis légèrement perturbés évoluent spontanément vers un nouvel état, lui aussi critique. Ce changement implique des modifications du système, celles-ci étant caractérisées par leurs amplitudes arbitrairement grandes.

L'exemple typique est celui de l'avalanche sur un tas de sable. Le tas de sable est initialement stable et la perturbation revient à ajouter un grain et à le laisser tomber sur le tas. Le tas évolue alors vers un nouvel état stable, par le biais d'un éboulement naissant à l'endroit où le grain est tombé. Le fait que la taille de cette avalanche soit arbitraire est caractéristique des systèmes *SOC*.

Depuis ce premier pas, plusieurs physiciens et biologistes ont reconnu ces caractéristiques dans des systèmes naturels, et, depuis lors, la famille *SOC* ne cesse de s'agran-

dir (cf. [Tan93] à titre d'exemple) ; d'ailleurs, ce domaine de recherche donne lieu depuis quelques années à de nombreuses publications [Bak97, Dur97, Tur97, Jen98]. Ces phénomènes sont particulièrement étudiés en physique des surfaces [BS95], en sismologie [BT98], dans l'étude de la dissipation dans les plasmas [CNLD96], des éruptions solaires [LH91], des superconducteurs [WS93] et, bien entendu, des systèmes granulaires comme les dunes, les silos à grains, les agrégations de molécules et beaucoup d'autres.

1.3 Le *Chip Firing Game* (*CFG*)

L'essence des phénomènes énoncés précédemment est capturée par un modèle largement utilisé en théorie des jeux et en combinatoire : le ***Chip Firing Game*** (*CFG*). Il s'agit d'un modèle permettant de décrire et prévoir l'évolution de systèmes granulaires. Il consiste à placer des jetons (grains) sur les sommets d'un graphe (orienté ou non) et à déplacer un grain du sommet s vers chacun de ses sommets adjacents, quand le nombre de ses jetons dépasse un seuil fixé.

Soit un graphe orienté $G = (V, E)$ pour lequel on associe un seuil δ_v et une charge $c(v)$ à chaque sommet $v \in V$. On assimile souvent $c(v)$ à un nombre de grains (ou de jetons) qui seraient stockés en v. Le jeu est alors le suivant : si $v \in V$ contient plus de δ_v grains alors il en expulse δ_v, c'est-à-dire que sa charge diminue de δ_v et celle de chacun de ses successeurs augmente de $\frac{\delta_v}{\sigma_v}$ où σ_v désigne le nombre de successeurs de v. On fixe de manière générale $\delta_v = \sigma_v$ sauf si $\sigma_v = 0$ auquel cas $\delta_v = \infty$ (v est alors un puits) (cf. la Figure 1 pour un exemple de *CFG*).

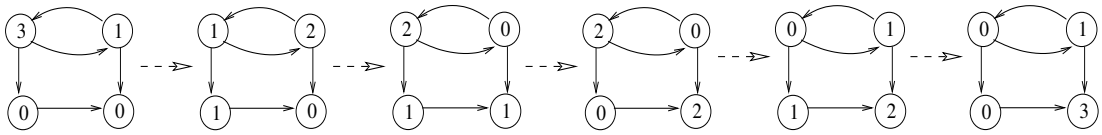


FIG. 1 – Un chemin d'un *Chip Firing Game*.

La figure précédente montre l'un des "chemins" possibles amenant à une stabilisation du modèle. Cependant, il n'existe pas, la plupart du temps, un unique chemin pour atteindre une configuration d'équilibre (si celle-ci existe) à partir de laquelle aucun mouvement de grains n'est plus possible. La Figure 2 en est une illustration.

La stabilisation du modèle n'est en effet pas obligatoire. Les *CFG* convergent vers un état d'équilibre s'ils vérifient certaines propriétés (cf. [Eri93]).

Si on incrémente la charge d'un des sommets $v \in V$ (on ajoute un grain à v) à partir d'un état critique, on induit éventuellement un rééquilibrage des charges de proche en proche. Lors de ces rééquilibrages, un nombre arbitraire de sommets peut être affecté. Pour mieux comprendre, à titre d'illustration, nous pouvons nous intéresser au cas où

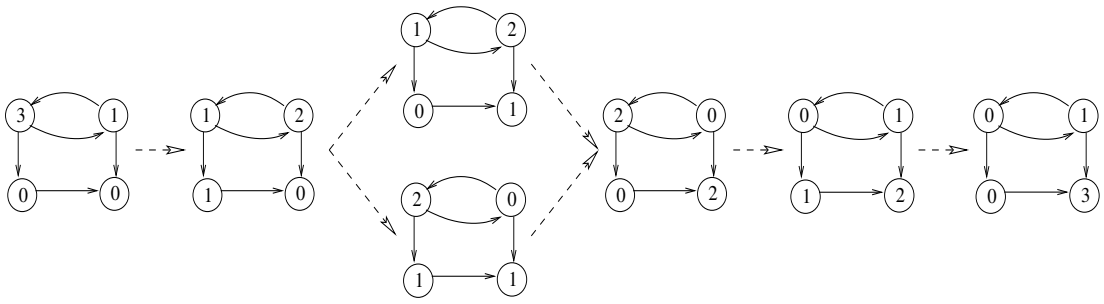


FIG. 2 – L'ensemble des chemins d'un *Chip Firing Game*.

$c(\nu) = 0$ et $c(v) = \delta_v$ *pour tout* $v \in V \backslash \{\nu\}$. Si on ajoute successivement $\delta_\nu - 1$ grains sur ν, le seul sommet affecté est ν. On reste par conséquent dans un état stable. Mais si on ajoute un grain de plus, tous les sommets sont affectés de proche en proche (si G est connexe) par propagation. On voit alors qu'une telle propagation peut être de taille arbitraire suivant l'état initial du modèle, et qu'elle est toujours provoquée par l'ajout d'un grain. Ce phénomène constitue une **avalanche**.

Maintenant que le cadre général du domaine d'étude est défini, nous allons introduire les principales recherches réalisées au LIAFA sur le *SPM*, qui est une spécialisation des *CFG*.

1.4 Le *Sand Pile Model* bi-dimensionnel (*SPM*)

1.4.1 Les premières définitions

Ce modèle consiste à prendre comme graphe sous-jacent un graphe $G = (V, E)$ filiforme, c'est-à-dire une chaîne, orienté de longueur infinie (à droite) tel que : $V = \mathbb{N}$, $E = \{(i, i+1),\ \forall i \in \mathbb{N}\}$. Plus précisément, ce modèle est strictement équivalent au suivant : on considère une suite infinie de colonnes, chacune contenant un empilement vertical de grains. Initialement, la première colonne contient les n grains et toutes les autres sont "vides". On note $d(c)$ la différence de hauteur entre la colonne numéro c et sa voisine de droite. On a alors la caractéristique suivante : si $d(c)$ excède 2 (c'est-à-dire $d(c) \geq 2$) alors un grain tombe de c sur sa voisine de droite notée c'. Ainsi, $d(c)$ diminue de 1 alors que $d(c')$ augmente de 1 (cf. la Figure 3 (a et b)). Comme on l'a anticipé dans l'introduction, il y a un lien entre ce modèle et les partitions d'entiers.

Définition 1.1 (partition) *: Une partition d'un entier positif n est une suite non croissante de n entiers positifs $a = (a_1, ..., a_n)$ tels que $\sum_{i=1}^{n} a_i = n$. Les a_i sont appelés les composantes de la partition a. Par abus de notation, nous pouvons désigner par a la partie initiale contenant toutes les composantes strictement positives de la partition a. Par exemple, la partition (4,2,1,0,0,0,0) de l'entier 7 sera représentée par (4,2,1).*

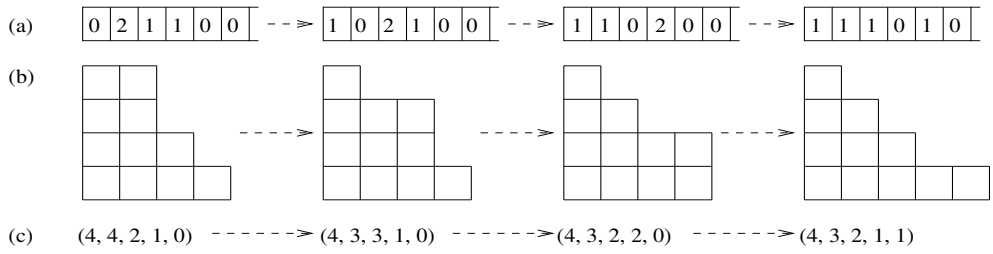


FIG. 3 – Trois manières différentes de voir un *SPM*.

Le modèle *SPM* est conservatif. Le nombre total de grains est en effet conservé par les transitions. On peut donc, en codant une pile par un n-uplet dont chaque composante est le nombre de grains dans la colonne correspondante, assimiler une configuration de la pile à une **partition** du nombre total de grain (cf. la Figure 3 (c)).

La section qui suit va présenter "l'état de l'art" concernant le modèle *SPM* au moment où j'ai commencé mon stage.

1.4.2 Les résultats connus et utiles au projet

Le modèle qui nous intéresse ici est le *SPM* séquentiel, c'est-à-dire qu'un seul grain tombe à chaque étape. Il est utile de le préciser car un autre modèle existe, le modèle en parallèle, qui permet de faire tomber tous les grains qui le peuvent à chaque étape.

Règle de base : La configuration correspondant à la partition $(n_1, n_2, ..., n_i, n_{i+1}, ...)$ peut évoluer en la configuration représentée par la partition $(n_1, n_2, ..., n_i - 1, n_{i+1} + 1, ...)$ *si et seulement si* $n_i - n_{i+1} \geq 2$. Dans ce cas, nous écrirons $(n_1, n_2, ..., n_i, n_{i+1}, ...) \longrightarrow (n_1, n_2, ..., n_i - 1, n_{i+1} + 1, ...)$.

Dans d'autres variantes, on peut modifier ce seuil, ici égal à deux, si on le souhaite. On peut de plus ajouter d'autres règles comme celle du **glissement** (cf. la Figure 4 (b)), de longueur bornée [GMP02] ou non [Bri73, GMP02].

Règle du glissement : La configuration correspondant à la partition $(n_1, n_2, ...)$ peut évoluer en la configuration représentée par la partition $(n_1, n_2, ..., n_{i-1} - 1, n_i, n_{i+1}, ..., n_{i+k}, n_{i+k+1} + 1, ...)$ *si et seulement si* $n_{i-1} = n_i + 1, n_i = n_{i+1} = ... = n_{i+k}$ *et* $n_{i+k+1} = n_{i+k} - 1$. Dans ce cas, nous écrirons $(n_1, n_2, ...) \longrightarrow (n_1, n_2, ..., n_{i-1} - 1, n_i, n_{i+1}, ..., n_{i+k}, n_{i+k+1} + 1, ...)$

Dans [Bri73], Brylawski démontre que le *SPM* séquentiel soumis à la règle de base et à celle du glissement de longueur non bornée correspond à l'ensemble de toutes les partitions de n. Ceci rend ce modèle particulièrement intéressant mais ne permet plus de considérer cette variante comme un *CFG* particulier, en raison de la perte de la propriété de la localité des chutes.

Dans le projet qui m'a été confié, c'est-à-dire l'étude de la généralisation du *SPM* par l'ajout d'une dimension supplémentaire, cette règle de glissement n'est pas prise en compte.

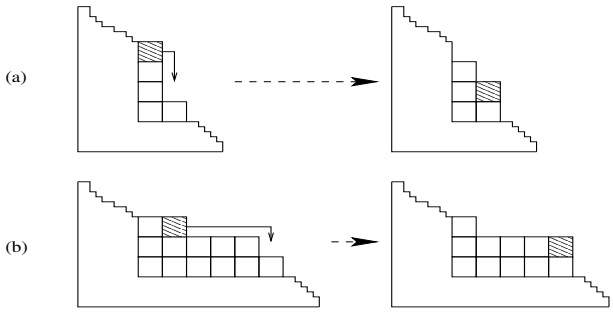


FIG. 4 – Les règles d'évolution du système.

Une approche fructueuse de cette étude est celle de la théorie des ordres. Il est donc utile ici de rappeler certaines définitions.

Définition 1.2 (ordre) *: Un ensemble ordonné est un couple $P = (S, \leq_P)$ où S est un ensemble et $\leq_P$ une relation d'ordre partiel sur S. L'ordre partiel est une relation binaire définie sur S qui est réflexive, antisymétrique et transitive. On note $<_P$ l'ordre strict associé à $\leq_P$ (qui est donc égal à $\leq_P$ privé de ses couples de réflexivité).*

Définition 1.3 (treillis) *: Un ordre $P = (S, \leq_P)$ est un treillis si et seulement si, pour tout couple (x, y) de P, il existe un plus petit majorant (borne supérieure) et un plus grand minorant (borne inférieure) communs à x et y.*

Le *SPM* séquentiel avec pour unique règle la règle de base permet d'obtenir un sous-ensemble propre de l'ensemble de toutes les partitions de l'entier n (le nombre total de grains). Cet ensemble de partitions, muni d'une relation d'**ordre** induite par la règle de base ($x > y$ *ssi* $x \longrightarrow y$ *ou* $\exists z$ *tel que* $x \longrightarrow z$ *et* $z > y$) a une structure de **treillis** [GK93, GMP02].

Remarque 1.4 *: Notons que l'ordre induit par le SPM séquentiel muni de la règle de base est isomorphe à l'ordre de dominance des partitions.*
Une partition $\alpha = (a_1, a_2, \ldots, a_n)$ est plus grande que la partition $\beta = (b_1, b_2, \ldots, b_n)$ selon l'ordre de dominance si $\sum_{i=1}^{j} a_i \geq \sum_{i=1}^{j} b_i$ pour tout $j = 1, 2, \ldots, n$.

En conséquence du fait que l'ensemble des partitions correspondantes aux configurations du système $SPM(n)$ a une structure de treillis, il existe un unique minimum absolu. Aucune transition n'est possible à partir de cet élément ; il est donc l'unique point fixe du système.

On dispose d'une caractérisation des partitions de n qui correspondent aux configurations de *SPM*.

Théorème 1.5 *[LMMP01, GMP02] : Les conditions nécessaires et suffisantes pour qu'une partition s soit une configuration de* SPM *sont :*

- *s ne contient pas de sous-séquence ..., p, p, p, ... ni de sous-séquence ..., $p, p, p-1, p-1$, ...*
- *entre deux séquences ..., p, p, ... et ..., q, q, ... consécutives, il y a au plus $p - q - 2$ valeurs.*

On dispose également d'une caractérisation des points fixes de *SPM*.

Théorème 1.6 *[GK93] : Si n est un nombre "triangulaire", c'est-à-dire de la forme $n = \frac{k(k+1)}{2}$ pour un k donné, alors le point fixe est la partition :*

$$(k, k-1, k-2, ..., 2, 1).$$

Si n n'est pas un nombre triangulaire, mais $n = \frac{k(k+1)}{2} + p$ avec $p < k + 1$, le point fixe est la partition :

$$(k, k-1, ..., p+1, p, p, p-1, ..., 2, 1).$$

Le principe de *SPM* est maintenant bien défini mais avant de conclure ces préliminaires et de passer à la présentation du travail réalisé pendant le stage, nous allons donner quelques définitions et notations utiles pour la suite. À la toute fin de ce chapitre sera présenté le treillis $SPM(11)$ à titre d'exemple.

Définition 1.7 (escalier) *: Dans une pile de sable $s = (s_1, ..., s_k)$, la position i est appelée escalier si $s_i - s_{i+1} = 1$. De même, un intervalle $[i, j]$ est appelé escalier si k est un escalier pour tout $k \in [i, j]$ (cf. la Figure 5).*

Définition 1.8 (plateau) *: Dans une pile de sable $s = (s_1, ..., s_k)$, la position i est appelée plateau si $s_i = s_{i+1}$. De même, un intervalle $[i, j]$ est appelé plateau si k est un plateau pour tout $k \in [i, j]$ (cf. la Figure 5).*

Définition 1.9 (falaise) *: Dans une pile de sable $s = (s_1, ..., s_k)$, la position i est appelée falaise si $s_i - s_{i+1} \geq 2$ (cf. la Figure 5).*

On peut donner une définition alternative de la règle d'évolution du système en utilisant ces dernières définitions.

Définition 1.10 (transition) *: Les piles de sable évoluent selon la règle suivante :*

$$(s_1, ..., s_i, s_{i+1}, ..., s_k) \xrightarrow{i} (s_1, ..., s_i - 1, s_{i+1} + 1, ..., s_k)$$

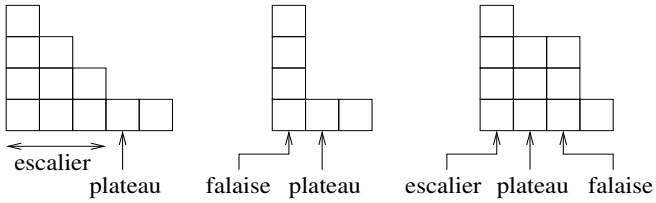


FIG. 5 – Exemples “d’architectures” d’un *SPM*.

si et seulement si i est une falaise. En d’autres termes, un grain peut chuter seulement s’il se trouve au bord d’une falaise. Une telle chute est appellée transition et est notée $\xrightarrow{i}$ où i est le numéro de la colonne de laquelle le grain tombe (cf. la Figure 6).

Définition 1.11 (successeur) *: Soient s et $s' \in SPM(n)$. s' est un successeur de s si et seulement s’il existe au moins un i tel que $s \xrightarrow{i} s'$. On note $Succ(s)$ l’ensemble des successeurs de s (cf. la Figure 6).*

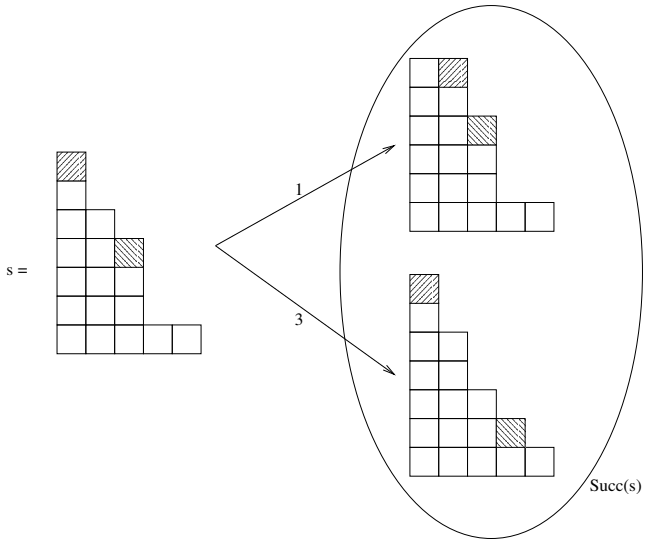


FIG. 6 – Transitions et successeurs

Notation 1.12 (*SPM(n)*) *: On désignera par $SPM(n)$ le treillis obtenu à partir de la partition (n,0,0,...) avec la règle de base (cf. SPM(11) de la Figure 7).*

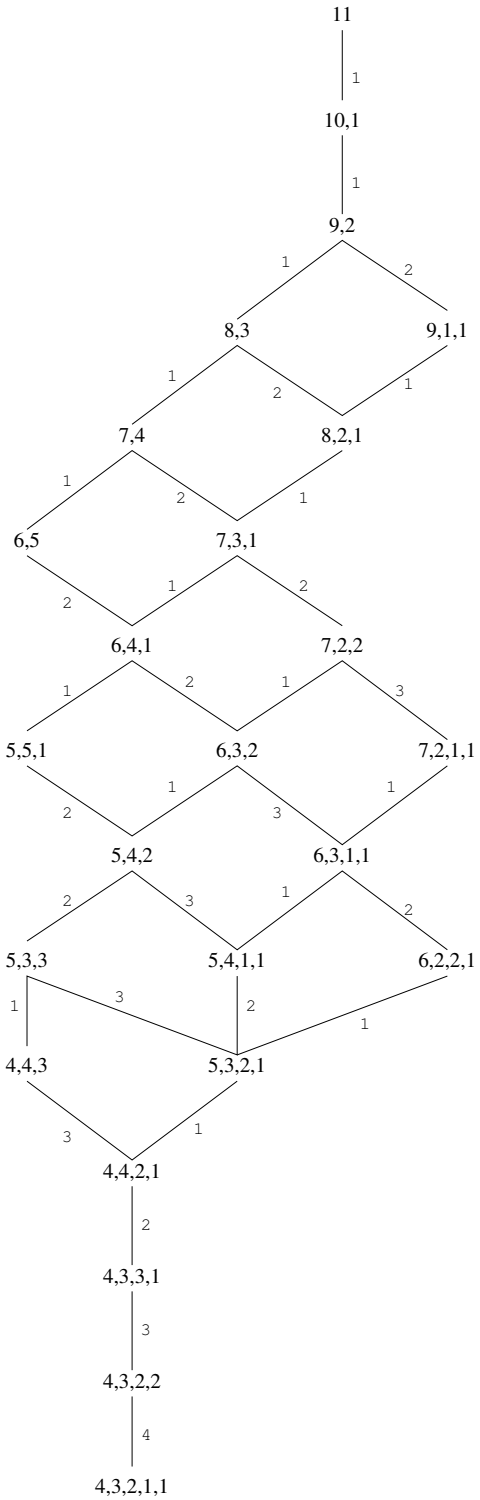

11
1
10,1
1
9,2
1
2
8,3
9,1,1
1
2
1
7,4
8,2,1
1
2
1
6,5
7,3,1
2
1
2
6,4,1
7,2,2
1
2
1
3
5,5,1
6,3,2
7,2,1,1
2
1
3
1
5,4,2
6,3,1,1
2
3
1
2
5,3,3
5,4,1,1
6,2,2,1
1
3
2
1
4,4,3
5,3,2,1
3
1
4,4,2,1
2
4,3,3,1
3
4,3,2,2
4
4,3,2,1,1

FIG. 7 – SPM(11)

2 Ajouter une nouvelle dimension : le modèle *BSPM*

Quoi de plus normal, à la suite des recherches effectuées sur *SPM*, que de s'intéresser au cas tri-dimensionnel ? Le monde qui nous entoure n'étant pas limité à deux dimensions, il paraît logique de vouloir se rapprocher au maximum de la réalité. C'est donc sur ce thème que mon stage s'est déroulé. Ainsi, dans ce chapitre, nous nous attacherons à exposer les recherches effectuées au cours de ces quatre mois. Il s'agit ainsi de la partie théorique du rapport dans laquelle nous mettrons en évidence l'existence de plusieurs modèles avant de justifier le choix de la généralisation adoptée. Ensuite, nous expliquerons de manière détaillée les deux différentes approches du problème auxquelles nous nous sommes interessés, l'une probabiliste et l'autre combinatoire.

2.1 Plusieurs généralisations possibles

Lorsque MM. Bak, Tang et Wiesenfeld ont introduit le modèle du *sand pile* pour illustrer le phénomène d'auto-organisation critique, dans les années 80, ils en ont donné une représentation sous la forme d'un automate cellulaire dont les cellules sont basées sur des grilles bi-dimensionnelles. Cependant, avant de nous attacher au modèle sur lequel s'est basé le projet de stage, nous énoncerons les principes du modèle abélien (basé sur un automate cellulaire) et expliciterons pourquoi ce modèle n'est pas celui étudié.

2.1.1 Le modèle abélien

Définition 2.1 (automate cellulaire) *: Un automate cellulaire est caractérisé par quatre propriétés :*

- *la géométrie des cellules. On choisit en général un réseau à une ou deux dimensions constitué par la juxtaposition de carrés identiques,*
- *dans un réseau donné, il faut préciser comment le voisinage d'une cellule modifie l'évolution de cette dernière. Dans le cas des réseaux bi-dimensionnels, deux types de voisinages ont principalement été étudiés : le voisinage de Von Neumann dans lequel une cellule a quatre voisins (N, S, E et O) et le voisinage de Moore dans lequel on ajoute les quatre cellules diagonales (NO, NE, SE, SO),*
- *le nombre d'états possibles des cellules : on se limite en général à deux états, "vivant" et "mort".*
- *la principale source de variété dans l'univers des automates cellulaires est le grand nombre de règles que l'on peut adopter pour déterminer l'état suivant en fonction de l'état des cellules de son voisinage (si k est le nombre d'états possibles de chaque cellule et n le nombre de cellules de son voisinage, il existe k^{k^n} règles possibles).*

Avant de commencer à décrire l'automate cellulaire (cf. la Définition 2.1) associé au modèle de Bak, Tang et Wiesenfeld, il est utile de noter que l'aspect plus algébrique du

système a été introduit par Deepak Dhar et son équipe en 1989. C'est d'ailleurs de lui que provient la dénomination de *abelian sand pile*.

L'idée des automates cellulaires est à peu près contemporaine de celle des ordinateurs. Les premières recherches sont dues à John Von Neumann et Stanislas Ulam au début des années 50. L'objectif était de réaliser un système simple, capable de se reproduire à la manière d'un organisme vivant.

L'automate du tas de sable abélien se décrit de la manière suivante. Soit une grille de n cases par p (une matrice $\mathcal{M}_{n\times p}$). Chaque élément $\mathcal{M}[i,j]$ de la grille représente une colonne de la pile de sable et contient en réalité le nombre de grains présents dans cette colonne. Il s'agit par conséquent d'une vue "aérienne" du tas de sable. La Figure 8 donne une illustration simple d'un tas de sable abélien sur une grille 2×2.

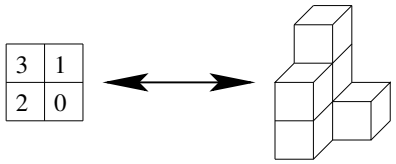


FIG. 8 – Une grille 2×2 d'un tas de sable abélien

On appelle configuration sur cette grille tout ensemble de nombres sur les cases de la grille. La Figure 9 présente la configuration $(3,2,1,3;0,3,3,1;1,4,2,0)$ (où chaque point virgule désigne une fin de ligne). La règle d'éboulement du tas de sable est la suivante : si une cellule contient quatre grains de sable au moins, elle s'éboule et perd quatre grains en en donnant un à chacune de ses voisines. Si cette cellule est sur le bord de la grille, alors les grains de sable qui sortent de la grille sont "perdus".

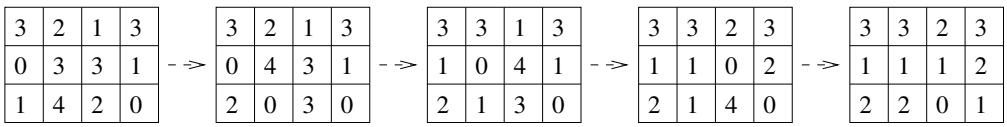


FIG. 9 – Évolution d'un tas de sable abélien.

On se rend compte assez rapidement, en regardant ce modèle, que sa règle d'éboulement n'est pas la plus proche possible de la réalité. En effet, un cas particulier (qui n'est pas unique) est celui de l'éboulement d'une colonne de quatre grains qui a pour voisines, au sens du voisinage de Von Neumann (cf. la Définition 2.1), uniquement des colonnes de trois grains. Dans ce cas bien spécial, si l'on décompose la transition effectuée en "sous-transitions", on se retrouve face à des états irréalisables car l'éboulement se caractériserait par des remontées de grains, ce qui est contradictoire et paraît quelque peu inconcevable (cf. la Figure 10).

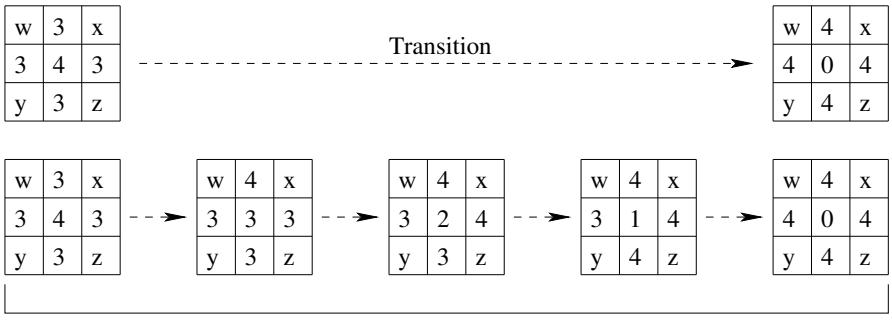


FIG. 10 – Décomposition d'un cas particulier.

Par ailleurs, le fait que l'éboulement ait lieu dans les quatre directions (Nord, Sud, Est et Ouest) en même temps paraît là encore assez étonant. Cela est néan-moins possible en environnement fermé, comme un silo à grains. Toutefois, forcer l'avalanche dans quatre orientations opposées semble ne pas correspondre aux conditions du milieu naturel que le système est censé représenter. À titre d'exemple, on peut avancer que le vent est un phénomène physique capable de provoquer la transformation d'une dune. Or, il est excessivement rare, voire impossible, de voir le vent souffler au même endroit et au même moment dans quatre directions différentes.

2.1.2 Le modèle étudié

Les travaux de recherche effectués par les chercheurs du LIAFA à ce sujet se sont fondés sur un autre modèle, issu d'une extension assez intuitive de *SPM*, en essayant de représenter plus fidèlement ce qui se passe dans la réalité.

En effet, dans ce nouveau modèle, chaque configuration est une distribution de grains de sable sur une grille rectangulaire immergée dans un quart de plan cartésien discret dont l'axe des abscisses est orienté vers l'Est et celui des ordonnées vers le Sud. La "limitation" de l'espace à un quart de plan n'est pas restrictive car elle peut être facilement étendue par symétrie aux autres directions. Nous noterons ce modèle par l'acronyme $BSPM$, où le B fait référence à la grille bi-dimensionnelle sous-jacente.

Il est possible d'envisager plusieurs manières de généraliser la règle d'évolution des $SPM(n)$ à ce système. Toutefois, il a été montré de façon expérimentale que l'ordre induit sur l'ensemble des configurations ne détermine pas de treillis dans aucune des généralisations qui semblent les plus naturelles. En particulier, il n'existe dans aucune des généralisations considérées une unique configuration stable, c'est-à-dire un unique point fixe (i.e. une configuration sur laquelle on ne peut appliquer aucune règle de transition), mais plusieurs. La Figure 11 illustre le schéma général de l'ordre induit d'un *SPM* sur une grille bi-dimensionnelle en comparaison à celui d'un *SPM* sur une chaîne uni-dimensionnelle.

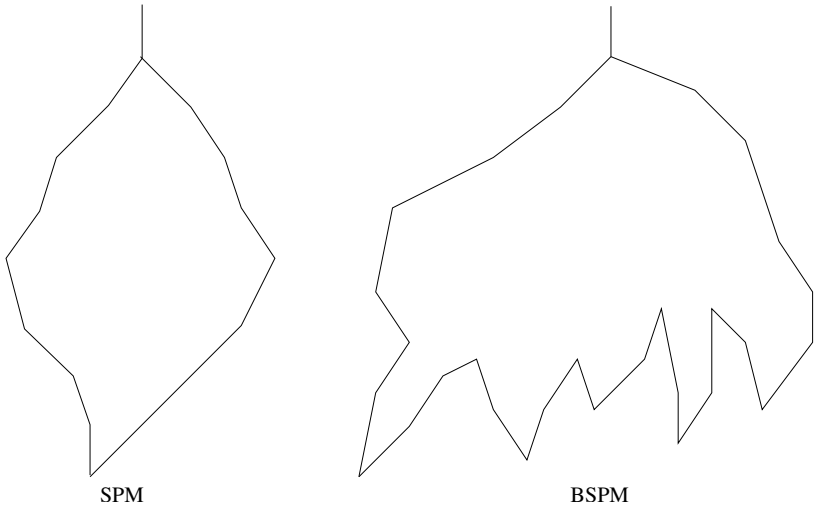


FIG. 11 – Schéma général de *SPM* sur une grille bi-dimensionnelle.

2.1.2.1 Quelques définitions

Tout d'abord, notons qu'à chaque fois que nous parlerons de décroissance au cours de ce rapport, il s'agira en fait de décroissance non stricte ou encore de non croissance.

Définition 2.2 (partition plane) *: Une partition plane $\mathcal{A}$ d'un entier positif n est une matrice telle que chaque ligne et chaque colonne est une suite décroissante d'entiers positifs et telle que $\sum_{i=1,j=1}^{n} \mathcal{A}[i,j] = n$. Les $\mathcal{A}[i,j]$ sont appelés les composantes de la partition $\mathcal{A}$. Par abus de notation, nous pouvons désigner par $\mathcal{A}$ la partie contenant toutes les composantes strictement positives de la partition $\mathcal{A}$. Par exemple, la partition plane* $\begin{pmatrix} 3 & 2 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ *sera représentée dans certains cas par* $\begin{matrix} 3 & 2 & 1 \\ 1 & & \end{matrix}$ *.*

Les règles d'évolution du système qu'on a retenues sont soumises aux deux hypothèses suivantes :

- la distribution des grains est toujours telle que les hauteurs des piles de sable sur les sommets de la grille sont décroissantes du Nord au Sud et d'Ouest en Est,
- les règles de mouvement sont les suivantes : au début, il y a une seule pile de sable que l'on peut supposer placée dans le sommet de coordonnées (1,1) de la grille ; à chaque instant, un grain de n'importe quelle pile non vide peut tomber vers le Sud ou vers l'Est à condition que la configuration obtenue reste toujours décroissante dans les deux directions Nord-Sud et Ouest-Est.

Par conséquent, comme le nombre total des grains de sable du tas est conservé par les mouvements, on peut assimiler chaque configuration à une **partition plane** (cf. la

Définition 2.2) du nombre total de grains dont chaque composante est le nombre de grains dans la pile correspondante.

De même que pour le cas *SPM*, nous utilisons les notions d'escaliers, falaises et plateaux. Leurs définitions restent presque identiques (et ne seront donc pas formalisées) à la différence près que ces notions ne sont pas uniquement relatives à une position mais aussi à une direction parmi Nord-Sud et Ouest-Est.

Il est toutefois utile de définir formellement ce qui caractérise les transitions de *BSPM* en différenciant les transitions Est des transitions Sud, ce qui est fait par la définition suivante.

Définition 2.3 (transition Est) *: Soit α une configuration et $\mathcal{A}$ la partition plane correspondante. Les piles de sable évoluent par transition Est selon la règle suivante :*

$$\begin{pmatrix} .. & .. & .. & .. \\ .. & \mathcal{A}[i,j] & \mathcal{A}[i,j+1] & .. \\ .. & .. & .. & .. \end{pmatrix} \xrightarrow{(E,i,j)} \begin{pmatrix} .. & .. & .. & .. \\ .. & \mathcal{A}[i,j]-1 & \mathcal{A}[i,j+1]+1 & .. \\ .. & .. & .. & .. \end{pmatrix}$$

À cause de la condition de non croissance sur les lignes et sur les colonnes, une configuration doit respecter certaines conditions locales pour pouvoir y appliquer une transition Est. En effet, un grain peut passer de la position (i,j) à la position $(i,j+1)$ si et seulement si :

- $\mathcal{A}[i,j] - \mathcal{A}[i,j+1] \geq 2$,
- $\mathcal{A}[i-1,j+1] > \mathcal{A}[i,j+1]$,
- $\mathcal{A}[i,j] > \mathcal{A}[i+1,j]$.

En échangeant le rôle des indices de ligne et de colonne, on obtient une définition analogue pour les transitions Sud.

Le fait de conserver en permanence, au cours de l'évolution du système, la condition de non croissance sur les lignes et sur les colonnes, empêche que localement on puisse avoir des "remontées" des hauteurs des piles, ce qui paraît une contrainte parfaitement justifiable du point de vue physique et qui était l'un des principaux "reproches" faits au modèle abélien. Ainsi, ce modèle semble être un modèle tout à fait légitime pour l'étude de l'évolution des tas de sable en trois dimensions. De plus, le fait que toutes les configurations ainsi obtenues soient caractérisables par des partitions planes ajoute un nouvel aspect intéressant à ce modèle, la théorie des partitions planes étant un sujet bien développé en soi.

Notation 2.4 (*BSPM(n)*) *: On désignera par $BSPM(n)$ l'espace des configurations obtenues à partir de la partition plane $\begin{pmatrix} n & 0 & ... \\ 0 & 0 & ... \\ ... & ... & ... \end{pmatrix}$ (dite "configuration initiale") en appliquant les règles d'évolution correspondant aux transitions Est et Sud.*

Une notion importante dans ce contexte est celle d'**énergie**.

Définition 2.5 (énergie) *: Pour une configuration de tas de sable α, l'énergie $E(\alpha)$ se définit comme étant le nombre de transitions effectuées pour obtenir α depuis la configuration initiale.*

Si $\mathcal{A}$ est la partition plane correspondante à la configuration α, l'énergie $E(\alpha)$ peut être calculée par la formule suivante : $E(\alpha) = \sum_{i=1,j=1}^{n} \mathcal{A}[i,j](i+j-2)$.

Remarque 2.6 *: Il est utile de noter qu'une notion d'énergie peut aussi être définie dans le modèle SPM et que cette définition pour $BSPM$ en est une généralisation. Par ailleurs, une conséquence directe de la définition précédente est que les graphes de SPM et $BSPM$ sont "gradués", à savoir que tous les chemins menant à une même configuration sont de même longueur.*

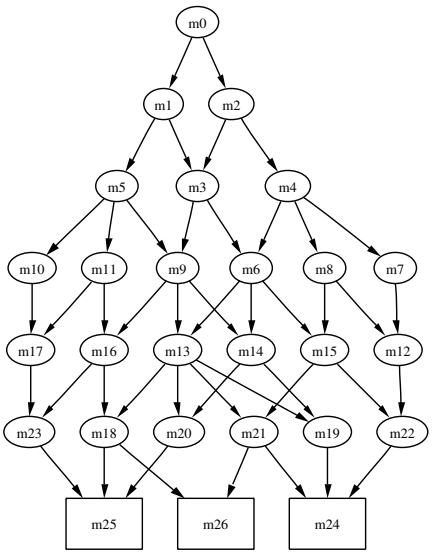


FIG. 12 – Energies ligne par ligne : $E(m0) = 0$, $E(m1) = E(m2) = 1, ...$

L'énergie correspond également au numéro de génération (au sens généalogique du terme) de la configuration à laquelle elle est associée. La Figure 12 donne une illustration de ce que nous appelons génération. Chaque ligne horizontale de sommets est une génération. Ainsi, tous les sommets d'une même ligne ont une énergie identique. Nous verrons plus tard que ce principe s'avère fort utile, notamment pour caractériser certains points fixes et pour une implémentation efficace du programme déterministe.

Contrairement au cas de *SPM*, ce n'est pas parce qu'une configuration contient une falaise qu'elle est forcément non stable. Ceci est une conséquence directe des propriétés de conservation de la décroissance des lignes et colonnes (i.e. conservation des partitions planes). Par conséquent, nous pouvons dans certains cas trouver des points fixes particuliers que nous appelons par abus de langage des **points fixes "bloquants"**.

Définition 2.7 (point fixe bloquant) *: Un point fixe bloquant est une configuration accessible arrivée à stabilité dont la partition plane $\mathcal{A}$ correspondante vérifie les propriétés suivantes : $\exists(i,j)$ tel que :*

- $\mathcal{A}[i,j] - \mathcal{A}[i,j+1] \geq 2$ *ou* $\mathcal{A}[i,j] - \mathcal{A}[i+1,j] \geq 2$ *(i.e. admet au moins une falaise) et*
- *si* $\mathcal{A}[i,j] - \mathcal{A}[i,j+1] \geq 2$ *alors*
  - $\mathcal{A}[i-1,j+1] \not> \mathcal{A}[i,j+1]$ *ou*
  - $\mathcal{A}[i,j] \not> \mathcal{A}[i+1,j]$
- *si* $\mathcal{A}[i,j] - \mathcal{A}[i+1,j] \geq 2$ *alors*
  - $\mathcal{A}[i+1,j-1] \not> \mathcal{A}[i+1,j]$ *ou*
  - $\mathcal{A}[i,j] \not> \mathcal{A}[i,j+1]$.

Il s'agit des configurations qui contiennent au moins une falaise mais dans lesquelles tout mouvement de grains entraînerait le non respect de la règle de décroissance sur les lignes ou sur les colonnes.

La Figure 13 (a) donne un exemple de point fixe bloquant.

Remarque 2.8 *: Il est utile de noter que, à partir d'un nombre de grains $n \geq 11$, il n'est pas rare de trouver des points fixes bloquants parmi les points fixes d'énergie minimale du système $BSPM(n)$.*

Un autre concept intéressant lorsque l'on veut caractériser les points fixes d'énergie minimale est celui des **points fixes pyramidaux**.

Définition 2.9 (nombre pyramidal) *: On dira qu'un entier n est pyramidal s'il existe un entier k tel que $n = \sum_{j=1}^{k} \sum_{i=1}^{j} i$. Par exemple :*
1
$1 + (1+2) = 4$
$1 + (1+2) + (1+2+3) = 10$
$1 + (1+2) + (1+2+3) + (1+2+3+4) = 20$
sont les quatre premiers nombres pyramidaux.

Définition 2.10 (point fixe pyramidal) *: Si n est un nombre pyramidal et k l'entier tel que $n = \sum_{j=1}^{k} \sum_{i=1}^{j} i$, on appelle point fixe pyramidal parfait la configuration α codée par la partition plane $\mathcal{A}$ telle que $\mathcal{A}[i,j] = k - i - j + 2$.*

Mis à part les points fixes bloquants, les points pyramidaux sont ceux qui minimisent la distance moyenne des grains de la case d'origine de coordonnées $(1,1)$. Ils sont donc les points fixes non bloquants d'énergie minimale. L'étude probabiliste qui sera détaillée dans le Paragraphe 2.2 a permis d'analyser la probabilité que le système a pour atteindre un point fixe d'énergie minimale quand on le laisse évoluer de façon aléatoire (par rapport

(a) bloquant (b) pyramidal

FIG. 13 – Points fixes bloquant et pyramidal de BSPM(13)

à la probabilité choisie). Toutefois, si n n'est pas un nombre pyramidal, on ne peut pas définir de point fixe pyramidal.

De plus, il a été utile de caractériser la **taille des partitions planes** en fonction du nombre de grains n du système.

Définition 2.11 (taille des partitions planes) *: La taille (maximale) des partitions planes (par brièveté, nous l'appellerons simplement "la taille des partitions planes") est l'entier τ correspondant au nombre de lignes de la plus petite matrice carrée qui peut contenir n'importe quelle partition plane correspondant à une configuration de $BSPM(n)$.*

Proposition 2.12 (taille des partitions planes) *: La taille τ des partitions planes de BSPM(n) est définie par :*

$$\tau = \begin{cases} \frac{-1+\sqrt{1+8n}}{2} & \textit{si le résultat obtenu est un nombre entier,} \\ \lfloor \frac{-1+\sqrt{1+8n}}{2} \rfloor + 1 & \textit{sinon.} \end{cases}$$

⌜ **Preuve** : La taille maximale que peut prendre un tas de sable au cours d'un éboulement est atteinte dans le cas où il s'agit d'une configuration de stabilité (i.e. un point fixe) ayant été obtenue soit uniquement par des transitions Est, soit uniquement par des transitions Sud.

Dans le cas d'un nombre de grains triangulaire, il s'agit donc d'une matrice de la forme suivante :

$$\begin{pmatrix} k & k-1 & ... & i & i-1 & ... & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... \end{pmatrix} \text{ ou } \begin{pmatrix} k & 0 & ... \\ k-1 & 0 & ... \\ ... & 0 & ... \\ i & 0 & ... \\ i-1 & 0 & ... \\ ... & 0 & ... \\ 2 & 0 & ... \\ 1 & 0 & ... \end{pmatrix}$$

La taille de la matrice, par conséquent déterminée par la longueur de la suite $u = (k, k-1, ..., i, i-1, ..., 2, 1)$, est égale à n. Comme $\sum_{i=1}^{k} u_i = n$, on a $n = \frac{k(k+1)}{2}$.

En conséquence, pour calculer la taille d'une partition plane, il suffit de résoudre l'équation du second degré suivante : $n = \frac{\tau(\tau+1)}{2}$ avec τ comme inconnue. Cette équation a deux solutions :

$\tau_1 = \frac{-1-\sqrt{1+8n}}{2} < 0$ et

$\tau_2 = \frac{-1+\sqrt{1+8n}}{2} > 0$.

Il est évident que seule la racine positive nous intéresse ici.

Par ailleurs, dans le cas où la racine n'est pas entière, ce qui se passe pour des nombres de grains non-triangulaires, on prend une marge de sécurité d'une case en ajoutant un à sa partie entière. Ceci est dû à la possibilité que u soit de la forme $(..., 1, 1)$ (i.e. se termine par deux 1). La marge de sécurité ne peut excéder une case car le motif p, p, p est interdit dans une configuration de *SPM(n)*.

⌟

2.1.2.2 Les aspects du problème

L'un des grands intérêts de ce thème de recherche est qu'il admet un grand nombre d'approches différentes. En effet, il suscite autant de questions chez les physiciens et biologistes, que chez les mathématiciens et informaticiens.

Ainsi, le physicien s'attachera plutôt à déterminer les modèles qui représentent le plus fidèlement la réalité, à vérifier qu'un modèle donné amène à de bonnes illustrations des phénomènes physiques et naturels existants. Le mathématicien sera, lui, plutôt attiré par des problèmes heuristiques, comme proposer des conjectures sur les propriétés mathématiques par le biais de résultats calculatoires, ou encore d'autres liés à la combinatoire des partitions planes et des ordres ainsi obtenus. Enfin, l'informaticien se situera entre les deux. Il essaiera de trouver des méthodes et procédés fiables et efficaces pour résoudre ces divers problèmes. Il étudiera par exemple des codages alternatifs pour les tas de sable, développera des algorithmes rapides pour le calcul des configurations accessibles ou encore appliquera la théorie des pavages parce que la tombée d'un grain correspond à des opérations de *flip* de tuiles en losange.

Il est important de noter que chacun de ces angles de vue peut amener à faire avancer la recherche sur ce domaine. C'est d'ailleurs la raison pour laquelle, au cours de ce stage, nous ne nous sommes pas "cantonner" à une seule approche mais avons essayé de les combiner de manière à en tirer un maximum de résultats possibles.

2.2 Hasard et probabilités

2.2.1 Les raisons

Pour étudier l'évolution du système en cas d'avalanche en milieu naturel, qu'il s'agisse d'une avalanche en montagne ou d'une chute de grains sur une dune de sable en plein

milieu du Sahara, on n'a pas besoin de construire toutes les configurations du système *BSPM* associé.

En effet, si l'on considère le graphe associé au modèle, qui contient toutes les configurations que le système peut atteindre, un effondrement, quel qu'il soit, correspond à un chemin (une succession de sommets reliés deux à deux par des arêtes) allant de l'état initial à l'un des points fixes du modèle (cf. la Figure 14).

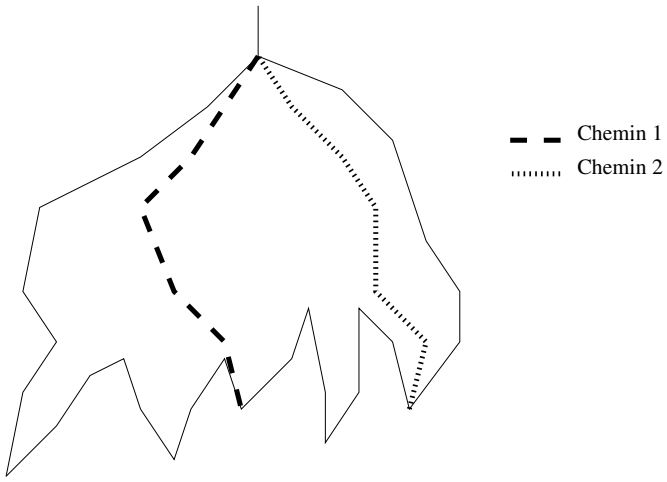


FIG. 14 – Exemples de chemins de BSPM.

Le choix du chemin est par conséquent essentiel. C'est lui qui va déterminer le déroulement de l'éboulement et aussi l'état final du système quand celui-ci se sera stabilisé. Or, il est évident que dans la nature, de nombreux facteurs interviennent dans ce choix. Afin de pouvoir matérialiser ces facteurs, nous utilisons des probabilités, ces dernières devant s'approcher au maximum des facteurs vraisemblables.

2.2.2 Les objectifs

Le but d'une telle étude probabiliste est d'analyser l'évolution du système sujet à des phénomènes physiques qui peuvent être paramétrés en utilisant des probabilités, et en conséquence avancer des prévisions par rapport à cette évolution dans certaines conditions.

Il est donc indispensable de développer des logiciels qui modélisent les différents critères possibles pour la définition de ces probabilités et qui, ensuite, simulent un nombre élevé d'évolutions "aléatoires" du système et tabulent les statistiques ainsi obtenues. Ceci permet en particulier de relever certaines irrégularités et d'avancer des hypothèses ou conjectures qui peuvent ensuite être confirmées (ou pas) par des preuves rigoureuses. En particulier, les données obtenues permettent de faire des liens entre la probabilité que le système se stabilise en une configuration donnée et l'énergie de cette dernière. Les

points fixes d'énergie minimale (les pyramidaux et ceux de forme proche à celles des pyramidaux) apparaissent en effet avec une fréquence supérieure.

En conséquence, il est d'abord nécessaire de définir précisément quelles sont les hypothèses de probabilités les plus intéressantes pour cet enjeu. Ceci n'est pas forcément une tâche facile puisqu'il faut pouvoir comparer des résultats provenant "d'hypothèses" différentes tout en conservant à l'esprit que plus ces "hypohèses" seront proches de la réalité, plus les résultats qu'on pourra en tirer auront de la valeur et auront par conséquent plus de chance de nous orienter vers la bonne voie.

Une fois la nature des probabilités fixée, il faut parvenir à simuler de façon efficace un très grand nombre de chemins pour faire en sorte que les statistiques obtenues soient les plus significatives possible. Pour ce faire, il est indispensable de réaliser un programme informatique rapide (qui sera détaillé dans la dernière partie du rapport).

2.2.3 Les différentes probabilités utilisées

Il existe plusieurs manières de concevoir les probabilités à mettre en œuvre. L'un des premiers critères à prendre en compte est celui de "l'objet" même sur lequel s'applique la probabilité. En effet, on peut privilégier l'un ou l'autre des critères. Par exemple, on peut choisir de se placer du point de vue des grains de sables, ou bien se dire que ce sont les piles elles-mêmes qui sont le plus probablement soumises aux phénomènes naturels. Le mieux, bien sûr, est de considérer ces deux points de vue et il peut d'ailleurs s'avérer utile de comparer les résultats auxquels ils mènent.

L'équiprobabilité des grains : La première probabilité qui est apparue comme logique a été de faire en sorte que chaque grain du système ait la même chance d'être choisi comme étant celui potentiellement intéressé par la prochaine transition. Ainsi, un grain situé au bas d'une pile (ne pouvant donc pas tomber) aura la même probabilité d'être choisi pour l'éboulement qu'un grain apte à bouger, c'est-à-dire un grain situé en haut d'une pile et respectant les propriétés caractéristiques d'une transition.

Plus formellement, dans un système $BSPM(n)$, chaque grain aura $\frac{1}{n}$ chance d'être choisi pour le déplacement.

Ce choix probabiliste est asynchrone, au sens où à chaque tirage aléatoire ne correspond pas forcément une transition du système. Les chercheurs en physique probabiliste conduisent cependant souvent leurs analyses avec des systèmes asynchrones.

L'équiprobabilité des piles : Du point de vue qui considère les piles et non les grains comme les "atomes" de ce système, il est normal de faire les mêmes hypothèses. Cela peut paraître plus en accord avec les phénomènes terrestres existants. Ainsi, chaque pile de sable aura autant de chance que les autres de s'ébouler. Dans un système $BSPM(n)$ où τ désigne la taille des partitions planes, chaque pile aura pour probabilité $\frac{1}{\tau^2}$ de tomber.

En réalité, nous pouvons noter que les propriétés de transition impliquent que l'avalanche, dans sa forme générale, ne peut en aucun cas dépasser les cases de la matrice

se trouvant à une distance supérieure à $n-1$ de la case (1,1), ce qui réduit de façon conséquente le nombre de piles à prendre en compte. Chaque pile apte à contenir des grains aura finalement $\frac{1}{\frac{\tau(\tau+1)}{2}-\tau}$ soit plus simplement $\frac{2}{\tau(\tau-1)}$ chances de s'ébouler.

La probabilité proportionnelle à la hauteur des piles : La théorie selon laquelle les piles les plus hautes sont plus propices à l'effondrement que les autres ne semble pas dénuée de sens. Dans le milieu naturel, cela peut s'illustrer par la force supérieure des vents en hauteur, ceci étant principalement une conséquence du fait que le vent a le temps d'être freiné par de nombreux obstacles à basse altitude. Cette probabilité a donc elle aussi été sélectionnée.

À chaque pile de sable est affectée la probabilité suivante : le quotient du nombre de grains qu'elle contient par le nombre de grains total du système, soit pour la pile de numéro i : $\frac{n_i}{n}$.

La probabilité relative aux différences de hauteur des piles : Ce dernier point présente la dernière probabilité adoptée, à savoir que la pile qui a le plus de chance de s'ébouler est celle qui contient la falaise la plus haute du système. Là encore, on peut considérer ce phénomène comme une combinaison des vents agissant sur le tas de sable et des forces physiques interagissant entre les piles de sable. Par conséquent, il est nécessaire ici de tenir compte non seulement de la pile, mais aussi de la direction (Est ou Sud). On assigne donc une probabilité non nulle à chaque mouvement faisable (mouvement respectant les propriétés des transitions Est ou Sud selon le cas) et non plus aux piles elles-mêmes. Ainsi, les objets à traiter seront des couples (α, β) dans lesquels α désigne le mouvement (Est ou Sud) à effectuer et β les coordonnées (i, j) de la pile en jeu.

Si l'on considère Σ_δ comme la somme totale de toutes les différences de hauteur (des mouvements faisables) existant entre les diverses piles du sytème et si l'on note $\delta^i_{(\alpha,\beta)}$ la différence de hauteur inhérente au mouvement i, la probabilité que le mouvement i soit celui réalisé est : $\frac{\delta^i_{(\alpha,\beta)}}{\Sigma_\delta}$.

2.2.4 Les principaux résultats obtenus

Grâce à notre programme visant à simuler de façon aléatoire un grand nombre de chemins pour générer des statistiques fiables, nous avons obtenu quelques résultats intéressants.

Les principaux résultats obtenus sont :

- Tout d'abord, nous avons pu confirmer l'hypothèse que les points fixes pyramidaux, pour un nombre total de grains du système pour lequel ils existent (4, 10, 20, ...), sont les points fixes "non-bloquants" d'énergie minimale.
- On ne peut en revanche pas affirmer que les points fixes pyramidaux sont toujours les plus probables à atteindre. Le contre-exemple suivant, qui provient de

statistiques engendrées pour cent mille chemins de $SPM(20)$, affiche suffisament d'écart entre les probabilités pour pouvoir infirmer cette conjecture. En effet, le point fixe $\begin{pmatrix} 3 & 3 & 2 & 1 \\ 3 & 2 & 1 & \\ 2 & 1 & 1 & \\ 1 & & & \end{pmatrix}$ est atteint dans $23,52\%$ des cas alors que le point fixe pyramidal ne l'est que dans $7,155\%$.

- Les études probabilistes ont de plus permis d'identifier des points fixes non bloquants et non pyramidaux d'énergie minimale. Nous pourrions les appeler "quasi-pyramidaux" car leur forme ressemble à celle des pyramidaux. Si cette nouvelle notion est admise, on peut émettre l'hypothèse que les points fixes les plus probables, quelque soit la probabilité utilisée, sont des points fixes "quasi-pyramidaux". Les résultats trouvés par voie heuristique pourront servir de point de départ pour la définition formelle, la classification et la caractérisation de ces points fixes.
- Les points fixes symétriques ont dans tous les cas des probabilités d'accès du même ordre de grandeur.

2.3 Combinatoire et déterminisme

2.3.1 Les raisons

L'approche probabiliste a bien sûr permis de donner quelques idées intéressantes sur la voie vers laquelle orienter les recherches. Cependant, les résultats obtenus ont une valeur expérimentale et ne permettent pas d'obtenir tous les renseignements souhaités, en particulier lorsqu'on se pose des questions liées à la caractérisation des configurations accessibles ou des points fixes. C'est pourquoi l'approche combinatoire est essentielle, même si elle est beaucoup plus lourde du point de vue de la complexité.

Cela revient non pas à arrêter les études à des instances de chemins calculées de manière aléatoire mais à être capable d'engendrer, simplement à partir d'un nombre n de grains, le graphe représentant l'espace de toutes les configurations issues de l'évolution du système $BSPM(n)$ à partir de sa configuration initiale. Ce graphe n'est rien d'autre qu'une représentation du diagramme de Hasse de l'ordre naturel introduit sur les configurations. Contrairement à l'approche probabiliste qui ne garantit pas, même en simulant un très grand nombre de chemins, d'obtenir à la fin l'ensemble de tous les points fixes du modèle, le déterminisme permet d'assurer que "rien n'a été oublié".

2.3.2 Les objectifs de recherche

Avant d'entamer la partie théorique de mon stage (la partie tournée vers la recherche), le premier objectif a été de développer un logiciel (dont l'implémentation sera le sujet du Paragraphe 3.2) capable de créer le graphe de l'ensemble de toutes les configurations de $BSPM(n)$ (avec en entrée le nombre de grains n contenus dans la configuration initiale)

pour avoir une panoplie d'exemples et de calculs effectués qui soit une base parfaitement fiable sur laquelle fonder nos recherches. Il est utile de noter à ce sujet que la taille de l'espace de toutes les configurations est relative au nombre de grains du système. On se rend compte que cette taille croît de manière exponentielle. Par conséquent, les propriétés essentielles que ce programme devait respecter était la rapidité (des structures de données et une algorithmique adaptées) et l'utilisation minimale de la mémoire. Ainsi, les deux aspects majeurs de la complexité (en espace et en temps) ont été les principaux aspects de cette réalisation.

Une fois la programmation achevée (des mises à jour ont malgré tout été effectuées par la suite), la partie "recherche" a pu débuter. C'est principalement ce moment du stage que j'attendais. Bien entendu, bien que les recherches sur le modèle *BSPM* aient avancé au cours de ces dernières années, de nombreuses questions restent encore sans réponse. Sur ce thème de recherche, le travail que l'on m'a donné était l'étude de l'une de ces questions : **comment déterminer si une configuration donnée est accessible selon le modèle *BSPM* sans recréer tout le chemin la séparant de la configuration initiale (cf. la Figure 15) ?** Il est certain que lorsqu'on lit cette question, le problème qu'elle pose paraît d'une grande simplicité. Pourtant, avant le début du stage, des chercheurs du LIAFA y avaient déjà réfléchi, par le biais de la théorie des ordres et l'étude des caractéristiques des partitions planes et n'avaient trouvé aucun résultat acceptable. Ceci n'est pas étonnant si l'on considère que déjà dans le cas bi-dimensionnel, la propriété qui caractérise les partitions qui sont des tas de sable est loin d'être facile.

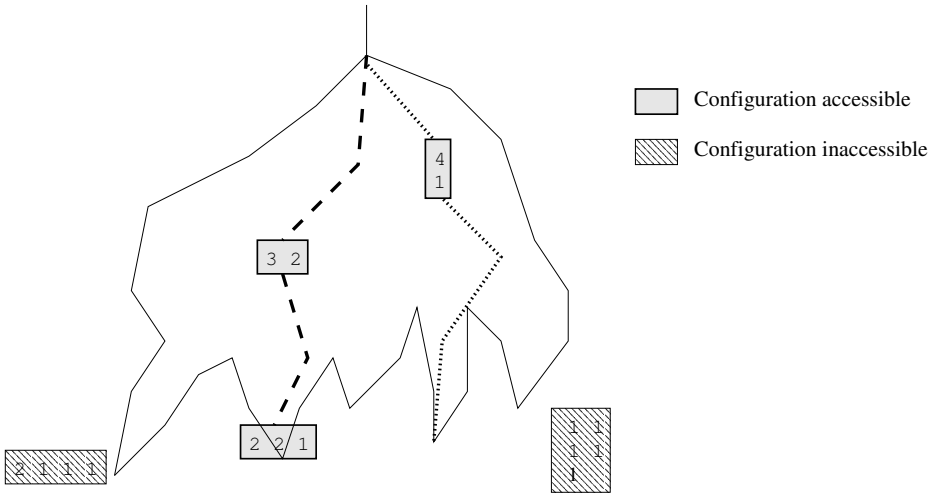


FIG. 15 – L'accessibilité des configurations.

Il s'avère qu'il s'agit donc d'un problème difficile et que sa résolution ferait faire une belle avancée aux recherches sur le sujet. Par conséquent, l'objectif majeur était de trouver une réponse correcte à cette question, ou en tout cas de parvenir à donner des réponses partielles. Une approche que nous avons développée pour la première fois

au cours du stage a été de créer et d'analyser des modèles mathématiques alternatifs de manière à trouver une caractérisation des configurations accessibles sur des objets canoniquement associés aux partitions planes plutôt que sur les partitions elles-mêmes.

Avant de nous intéresser à ces modèles alternatifs, nous allons donner le résultat essentiel provenant de l'étude des partitions planes.

2.3.3 L'étude des partitions planes

2.3.3.1 Un unique "motif" interdit

Définition 2.13 (motif interdit) *: On appelle motif tout polyomino connexe dont les cases sont remplies d'entiers dépendants d'un ou plusieurs paramètres. Un motif est dit interdit si, quelle que soit la valeur assignée à ces paramètres, aucune configuration du système contient le motif dans la partition plane qui lui est associée.*

Remarque 2.14 *Si une partition plane contient un motif interdit, il est impossible d'appliquer une transition inverse (et donc de faire remonter un grain) à une case appartenant au motif.*

Il a été démontré dans les recherches antérieures au stage qu'une condition nécessaire, induite des règles de transition, pour qu'une configuration soit une partition plane de $BSPM(n)$ est qu'elle ne doit pas contenir le "motif" $\begin{array}{|c|c|}\hline p & p \\ \hline p & p \\ \hline \end{array}$. En fait, on ne peut remonter aucun grain sans passer outre la règle de non croissance des lignes et colonnes. Il s'agit donc d'un **motif interdit**.

Au cours du stage, Ha Duong Phan et Dominique Rossin ont démontré qu'il s'agit du seul motif interdit.

Proposition 2.15 (unicité du motif interdit) *: Le modèle $BSPM$ admet un unique motif interdit* $\begin{array}{|c|c|}\hline p & p \\ \hline p & p \\ \hline \end{array}$.

⌜ **Preuve** : Supposons que la matrice $\mathcal{A}$ soit un motif interdit de taille minimale et que $\mathcal{A}$ ne contient pas $\begin{array}{|c|c|}\hline p & p \\ \hline p & p \\ \hline \end{array}$. De plus, admettons que $\mathcal{A}$ est composé de coefficients $\mathcal{A}[i,j]$ tels que $1 \leq i \leq k$, et $1 \leq j \leq l$.

Posons $p = \mathcal{A}[1,l]$. Comme $\mathcal{A}$ est un motif interdit, on ne peut remonter aucun grain à partir de $\mathcal{A}$. On ne peut donc remonter aucun grain de $\mathcal{A}[1,l]$ au nord. Ceci implique que $\mathcal{A}[2,l] = \mathcal{A}[1,l] = p$.

Par hypothèse, comme $\mathcal{A}$ ne contient pas $\begin{array}{|c|c|}\hline p & p \\ \hline p & p \\ \hline \end{array}$, on a $\mathcal{A}[1,l-1] \neq p$, c'est-à-dire que $\mathcal{A}[1,l-1] = q > p$.

Or, encore une fois, comme $\mathcal{A}$ est un motif interdit, on ne peut pas remonter un grain de $\mathcal{A}[1, l-1]$, donc $\mathcal{A}[2, l-1] - 1 = q$.

Et ainsi de suite, on a $\mathcal{A}[1,1] > \mathcal{A}[1,2] > ... > \mathcal{A}[1, l-1] > \mathcal{A}[1, l]$.

Une conséquence de la présence de l'escalier est que l'on peut dans ce cas, ajouter un grain à $\mathcal{A}[1, l]$ (à partir de sa colonne à droite), et remonter ce grain jusqu'à $\mathcal{A}[1, 1]$.

En continuant, on peut ajouter un grain à $\mathcal{A}[1, l]$, puis remonter jusqu'à $\mathcal{A}[1, 2]$...

Et ainsi de suite, on peut ajouter des grains de cette manière de telle façon que la première ligne ($\mathcal{A}[1,1]$,...,$\mathcal{A}[1, l]$) devienne une ligne $\mathcal{B}_1, ...\mathcal{B}_l$ avec des valeurs $\mathcal{B}_j$ arbitraires (assez grandes).

En procédant exactement pareil pour la deuxième puis les lignes suivantes, on en arrive à la conclusion que $\mathcal{A}$ n'est pas un motif interdit car on peut l'obtenir à partir d'une matrice suffisament grande.

⌟

2.3.3.2 Les sections interdites

La présence du motif interdit dans une configuration n'est en effet pas une condition nécessaire et pour qu'une configuration soit inaccessible. Nous avons pu remarquer, au cours des recherches, que d'autres formes de partitions planes ne présentant pas ce motif ne faisaient pas partie des partitions planes de $BSPM$.

Il existe en effet des sections de cases qui peuvent s'avérer acceptables dans des configurations de $BSPM$ si elles se trouvent dans certaines positions et qui peuvent aussi entrîner l'inaccessibilité d'une configuration si elles sont situées à d'autres positions. Dans ce deuxième cas, on dira que cette section est une "section interdite".

La section | p | p | p | est par exemple une section potentiellement interdite car elle est acceptable par $BSPM$ dans certains cas et pas d'en d'autres.

En effet, la configuration $2 \quad 1 \quad 1 \quad 1$ est inaccessible alors qu'en ajoutant une ligne comme suit, $\begin{matrix} 8 & 7 & 5 & 4 \\ 2 & 1 & 1 & 1 \end{matrix}$, elle devient accessible.

Le travail de recherche a donc consisté à essayer de trouver des conditions permettant de déterminer avec précision la présence des sections interdites.

2.3.4 Les modèles mathématiques alternatifs

Il est commun de représenter des objets tri-dimensionnels sur deux dimensions en utilisant des "courbes de niveaux" : on pense par exemple aux reliefs montagneux sur les cartes géographiques (cf. la Figure 16).

Une configuration contenant n grains possède n courbes de niveaux. On appellera k-ième ligne de niveau la ligne polygonale séparant la région du quart de plan incluant les cases qui contiennent un nombre de grains supérieur ou égal à k de la région incluant les cases qui contiennent un nombre de grains inférieur à k

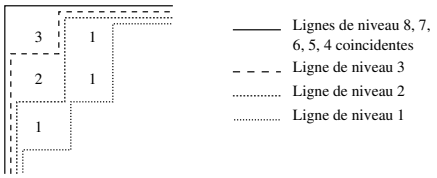


FIG. 16 – Une représentation des lignes de niveaux d'une configuration.

Les règles et définitions précédentes sur les partitions planes permettent d'admettre aisément deux propriétés relatives aux courbes de niveau :

- les lignes de niveau ne peuvent pas se croiser (conséquence directe des partitions planes),
- en choisissant opportunément leur sens de parcours , les courbes de niveau ne sont constituées que de segments ayant l'orientation Sud-Nord ou Ouest-Est. La Figure 17 suivante illustre une configuration impossible selon *BSPM* ne satisfaisant pas cette condition.

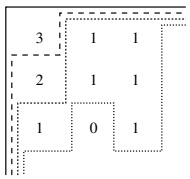


FIG. 17 – Un cas impossible.

Pour qu'une matrice représente une configuration accessible, il est bien entendu nécessaire mais pas suffisant que ses lignes de niveau satisfassent ces deux propriétés.

Il est utile que toutes les courbes de niveau aient la même longueur et que cette longueur soit la même pour toutes les partitions planes d'un même entier n. Pour cela, il suffit de placer le point de départ des courbes de niveau suffisament en bas, c'est à dire en position $((\tau+1), 0)$, ce qui donne des courbes de niveaux de longueur $2(\tau+1)$. Ceci est particulièrement utile pour la correspondance entre les courbes de niveau et les mots et chemins de Dyck que nous allons introduire maintenant.

Définition 2.16 (Mots de Dyck) *: Les mots de Dyck de premier ordre sont les mots engendrés par la grammaire suivante :*

$\mathcal{S} \rightarrow a\mathcal{S}b \mid \mathcal{S}\mathcal{S} \mid \varepsilon$.

Remarque 2.17 *Soit $\mathcal{D}_1^*$ l'ensemble des mots de Dyck de premier ordre et w un mot.*

$w \in \mathcal{D}_1^* \Longleftrightarrow |w|_a = |w|_b$ *et* $\forall u$ *préfixe de* w, $|u|_a \geq |u|_b$

Définition 2.18 (Chemins de Dyck) *: Les chemins de Dick ...*

Les deux modèles mathématiques suivants sont venus de manière assez intuitive car ils ne représentent que des combinaisons de mots de Dyck (sous diverses représentations) sous forme de matrice, chaque composante des combinaisons reproduisant une courbe de niveau.

2.3.4.1 Le modèle de l'ab-Matrice

La Figure 18 illustre comment on peut faire correspondre une configuration à un faisceau de n chemins de Dyck (ses courbes de niveaux pivotées de 45 degrés dans le sens des aiguilles d'une montre) et donc à une famille de n mots de Dyck, en utilisant la correspondance naturelle entre chemins et mots de Dyck.

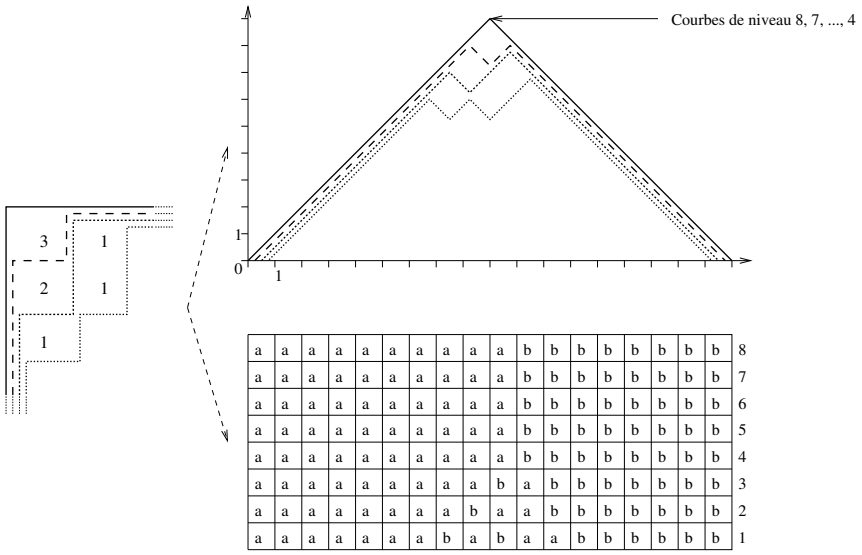


FIG. 18 – Chemins de Dyck et ab-Matrice.

La matrice de taille $n \times (2(\tau + 1))$ ainsi obtenue sera dite l'ab-Matrice de la configuration donnée.

2.3.4.2 Le modèle de la f-Matrice

Nous pouvons définir une autre matrice associée à une configuration α.
Pour ce faire, il faut considérer la fonction f définie par :

$$f(w) = |w|_a - |w|_b$$

Si w est un mot de Dyck de longueur m, on peut lui associer un vecteur $(f_1, f_2, ..., f_m)$, que nous appellerons f-vecteur et où chaque f_i est défini par :

$$f_i = f(w_i) \text{où } w_i \text{ est le préfixe de } w \text{ de longueur } i.$$

Pour une configuration α, on peut donc créer une matrice de taille $n \times (2(\tau+1))$ où la j-ième ligne est le f-vecteur associé au mot de Dyck codant la j-ième ligne de niveau.

Les f-vecteurs admettent les propriétés suivantes :

- $f_1 = 1$,
- $(f_{m-1}, f_m) = (1, 0)$,
- $|f_i - f_{i-1}| = 1$,
- $f_i \geq 0$ *pour tout* $i \in [1, ..., m]$.

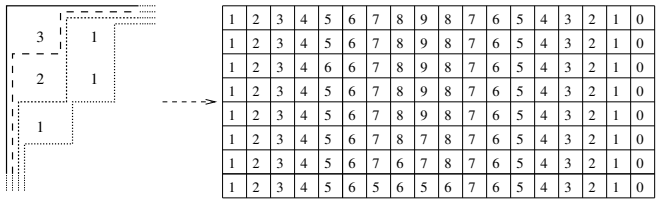


FIG. 19 – Exemple de f-Matrice.

Remarque 2.19 *Si $\mathcal{F}$ est la f-Matrice d'une configuration α, l'entier $\mathcal{F}[i,j]$ n'est rien d'autre que l'ordonnée du point d'abscisse j du chemin de Dyck correspondant à la i-ième courbe de niveau.*

2.3.5 Les recherches et les résultats obtenus

Dans cette section, nous allons présenter les différentes étapes que nous avons suivies pour arriver à identifier les propriétés qui caractérisent les partitions planes qui appartiennent à $BSPM(n)$ en utilisant les représentations alternatives que nous avons introduites. Ainsi, nous allons tout d'abord évoquer les principales caractéristiques trouvées des modèles mathématiques mis en œuvre.

2.3.5.1 Les principales propriétés des modèles

L'étude de ces modèles, qu'il s'agisse des ab-Matrices ou des f-Matrices, passe avant tout par trouver un maximum de propriétes les caractérisant.

Proposition 2.20 *Les colonnes des f-Matrices des configurations de $BSPM$ sont toujours décroissantes du haut vers le bas.*

⌜ **Preuve** : La propriété de décroissance des lignes et colonnes d'une partition plane implique que le chemin de Dyck correspondant à une ligne de niveau k ne peut pas se trouver au-dessus de celui correspondant à une ligne de niveau k' si $k < k'$, car les lignes de niveaux ne se croisent pas (cf. la Figure 20).

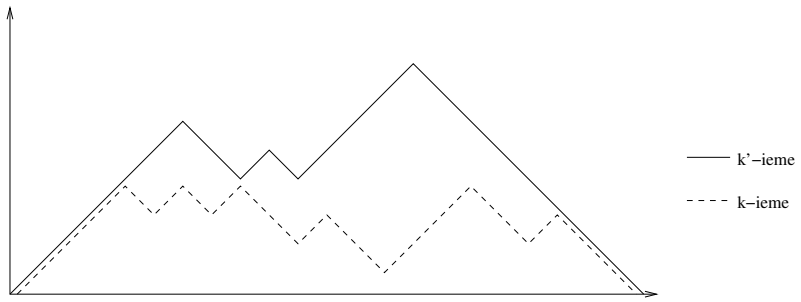


FIG. 20 – Non croisement des lignes de niveau.

⌟

Proposition 2.21 *Tous les entiers d'une f-Matrice se trouvant sur une même colonne ont la même parité.*

⌜ **Preuve** : Cette preuve est triviale car un entier $\mathcal{F}[i,j]$ est toujours obtenu en additionnant ou en soustrayant 1 à l'entier $\mathcal{F}[i,j-1]$ et $\mathcal{F}[i,1] = 1$ pour tout i. Les entiers d'une colonne ont donc parité opposée à celle des entiers se trouvant dans la colonne précédente.

⌟

Proposition 2.22 *Deux cases consécutives sur une même colonnes de la f-Matrice d'une configuration de $BSPM$ ne peuvent avoir une différence de quatre ou plus.*

⌜ **Preuve** : Si c'était le cas, cette différence marquerait la présence du carré $\begin{array}{|c|c|}\hline p & p \\ \hline p & p \\ \hline \end{array}$ qui est le motif interdit. En effet, une telle différence se schématise d'après la fonction $f(w)$ comme le montre la Figure 21 qui, par simple rotation de 45 degrés dans le sens contraire des aiguilles d'une montre, indique de façon certaine la présence de ce motif.

⌟

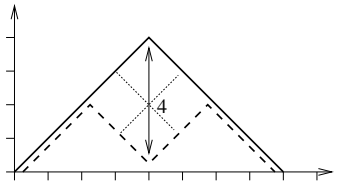


FIG. 21 – Lignes de niveau et motif interdit.

Proposition 2.23 *Pour tout* $i = 1, 2, ..., n-1$, *on a* $\mathcal{F}[i,j] - \mathcal{F}[i+1,j] = \begin{cases} 0 \\ 2 \end{cases}$.
Autrement dit, lorsqu'il y a décroissance stricte sur les colonnes d'une f*-Matrice, le terme de décroissance est exactement 2.*

⌜ **Preuve** : Cette proposition est une conséquence directe des Propositions 2.20, 2.21 et 2.22.

⌟

Il est également intéressant d'étudier la relation existant entre les transitions et les ab-Matrices et f-Matrices.

Regardons quel est l'effet d'une transition du système sur les lignes de niveau et donc sur les modèles mathématiques alternatifs. Pour ce faire, analysons tout d'abord l'effet d'une transition Est sur une ab-Matrice.

Une transition Est est fondée sur le schéma général de la Figure 22 qui montre deux cases côte-à-côte horizontalement (la case de gauche va donner un grain à la case droite). Pour que la règle de décroissance sur les lignes soit respectée, il est indispensable que ce qui se trouve au sud de la case de gauche contienne des valeurs inférieures à cette dernière et que ce qui se trouve au nord de la case de droite contienne des valeurs supérieures à celle-ci.

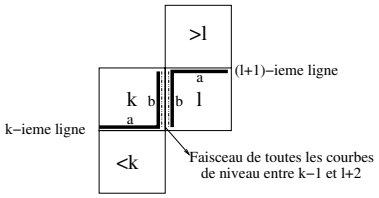


FIG. 22 – Schéma général de départ d'une transition Est.

Lorsqu'on effectue la transition vers l'Est, les lignes de niveau se replacent comme l'indique la Figure 23.

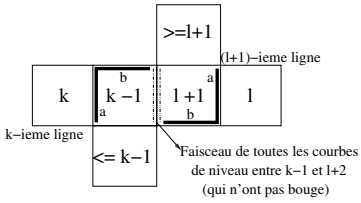


FIG. 23 – Schéma général de résultat d'une transition Est.

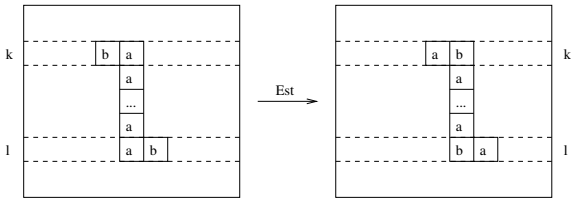


FIG. 24 – Transformation induite sur une ab-Matrice par une transition Est.

Par conséquent, au niveau de l'ab-Matrice, on a la transformation de la Figure 24.

On en déduit une proposition pour caractériser les transitions Est et par analogie, on en obtient une pour les transitions Sud.

Proposition 2.24 **(ab-Motifs d'une transition Est)** *: Toute configuration dont l'ab-Matrice contient au moins un motif*

b	a	$*$
$*$	a	$*$
$\vdots$	$\vdots$	$\vdots$
$*$	a	b

n'est pas une configuration stable. On peut lui appliquer une transition Est qui donnera une nouvelle configuration dans laquelle le motif précédent de l'ab-Matrice associée deviendra

a	b	$*$
$*$	a	$*$
$\vdots$	$\vdots$	$\vdots$
$*$	b	a

.

Proposition 2.25 **(ab-Motifs d'une transition Sud)** *: Toute configuration dont l'ab-Matrice contient au moins un motif*

$*$	b	a
$*$	b	$*$
$\vdots$	$\vdots$	$\vdots$
a	b	$*$

n'est pas une configuration stable. On peut lui appliquer une transition Sud qui donnera une nouvelle configuration dans laquelle le motif précédent de l'ab-Matrice associée deviendra

$*$	a	b
$*$	b	$*$
$\vdots$	$\vdots$	$\vdots$
b	a	$*$

.

De la même manière, nous pouvons étudier la géométrie des transitions sur les f-Matrices. La correspondance entre le schéma général de transition et les chemins de Dick est :

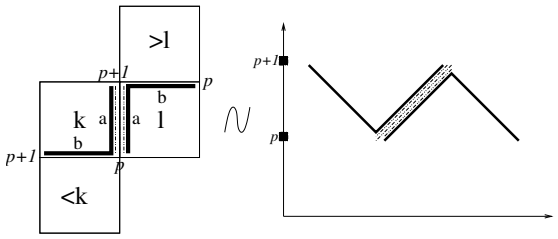


FIG. 25 – Les transitions Est et les chemins de Dyck.

Lorsqu'on effectue la transition vers l'Est, les lignes de niveau se replacent comme l'illustre la Figure suivante :

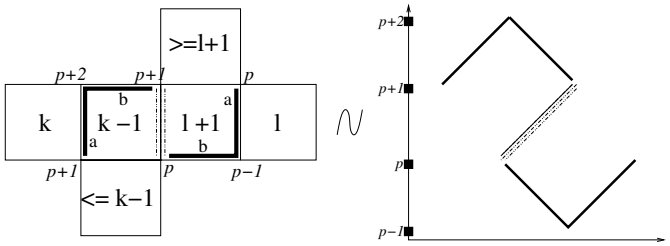


FIG. 26 – Les résultats de transitions Est et les chemins de Dyck.

Par conséquent, au niveau de la f-Matrice, on a la transformation présentée dans la Figure 27.

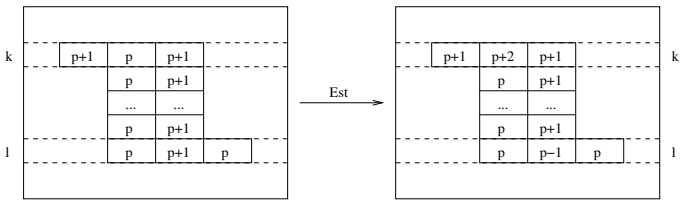


FIG. 27 – Transformation induite sur une f-Matrice par une transition Est.

Ainsi, par simple déduction, on obtient les propositions suivantes pour les f-Matrices. Certains coefficients sont indicés par les lettres correspondantes dans l'ab-Matrice pour faciliter la compréhension.

Proposition 2.26 (f-Motifs d'une transition Est) *: Toute configuration dont la f-Matrice contient au moins un motif*

$p+1$	$p_{(b)}$	$p+1_{(a)}$	$*$
$*$	p	$p+1_{(a)}$	$*$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$*$	p	$p+1_{(a)}$	$p_{(b)}$

n'est pas une configuration stable. On peut lui appliquer une transition Sud qui donnera une nouvelle configuration dans laquelle le motif précédent de la f-Matrice associée deviendra

$p+1$	$p+2_{(b)}$	$p+1_{(a)}$	$*$
$*$	p	$p+1_{(a)}$	$*$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$*$	p	$p-1_{(a)}$	$p_{(b)}$

.

Proposition 2.27 (f-Motifs d'une transition Sud) *: Toute configuration dont la f-Matrice contient au moins un motif*

$*$	p	$p-1_{(b)}$	$p_{(a)}$
$*$	p	$p-1_{(b)}$	$*$
...	...	...	...
$p-1$	$p_{(a)}$	$p-1_{(b)}$	$*$

n'est pas une configuration stable. On peut lui appliquer une transition Sud qui donnera une nouvelle configuration dans laquelle le motif précédent de la f-Matrice associée deviendra

$*$	p	$p+1_{(b)}$	$p_{(a)}$
$*$	p	$p-1_{(b)}$	$*$
...	...	...	...
$p-1$	$p-2_{(a)}$	$p-1_{(b)}$	$*$

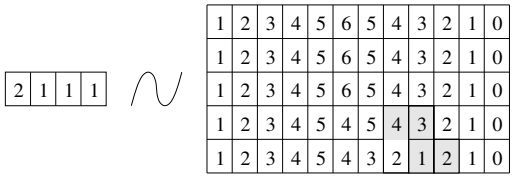
.

2.3.5.2 Des exemples de voies de recherche empruntées

Au cours de cette partie du stage, nous avons étudié successivement diverses pistes afin de trouver la solution au problème posé. Cependant, chacune des conjectures auxquelles nous avons pensé a été infirmée par les études réalisées et particulièrement par la découverte de contre-exemples. Dans ce paragraphe, nous donnerons un récapitulatif de quelques réflexions entreprises.

Conjecture 2.28 *: Une configuration est accessible si et seulement si elle ne contient pas le motif interdit et que sa f-Matrice contient au moins un motif de résultat de transition.*

En étudiant cette possibilité, on se rend vite compte qu'il s'agit là encore de conditions nécessaires mais pas suffisantes puisque cela permet non pas de prouver qu'une partition plane est une partition plane de $BSPM$ (i.e. qu'elle représente une configuration accessible depuis la configuration initiale) mais qu'il s'agit d'une partition plane atteignable (en un pas) à partir d'une autre qui, elle, peut ne pas être accessible. Un contre-exemple très simple prouve que cette condition n'est pas suffisante. Il s'agit d'une configuration inaccessible dont la f-Matrice contient un motif résultat de transition Est.



Conjecture 2.29 *: Une configuration est accessible si et seulement si elle ne contient pas le motif interdit, que ses ab-Matrice et f-Matrice contiennent chacune au moins un motif résultat de transition.*

Cette conjecture s'est avérée exacte pour les configurations directement inaccessibles, c'est-à-dire les configurations sur lesquelles on ne peut pas effectuer de transitions inverses d'un pas, telles que $2\quad 1\quad 1\quad 1$. Elle n'est toutefois pas valable pour les configurations qui sont créées en effectuant une transition à partir d'une configuration elle-même inaccessible. En effet, la configuration $2\quad 2\quad 1\quad 1$ qui peut être atteinte depuis la configuration inaccessible $2\quad 2\quad 2$ possède bien le motif résultat d'une transition Est à la fois dans son ab-Matrice et dans sa f-Matrice.

Il s'agit donc encore de propriétés nécessaires mais pas suffisantes.

En voyant que cette conjecture ne permettait pas de caractériser l'accessibilité d'une configuration, nous en avons émises de nouvelles en y ajoutant petit à petit les différentes propriétés que nous avons découvertes. Malheureusement, cela ne nous a pas amenés au résultat escompté. Nous sommes toujours arrivés à la même conclusion, c'est-à-dire que toutes les propriétés connues sont nécessaires mais pas suffisantes pour déterminer l'accessibilité.

Nous avons essayé par la suite de changer de "pistes" et de regarder attentivement ce qui semblait revenir dans les ab-Matrices et f-Matrices caractéristiques des configurations inaccessibles. En analysant diverses configurations accessibles et inaccessibles assez proches les unes des autres, nous avons remarqué que chacune des matrices ne correspondant pas à des partitions planes de $BSPM$ avait une ab-Matrice particulière. En

effet, cette dernière comportait soit un isolement d'un "a" au sein de trois colonnes ne comportant que des "b", soit un isolement d'un "b" au sein de trois colonne ne comportant que des "a". D'où la conjecture suivante :

Conjecture 2.30 *: Une configuration est accessible si et seulement si elle ne contient pas le motif interdit et que son ab-Matrice ne contient pas les "motifs d'isolement", à savoir*

b	*b*	*b*
b	*b*	*b*
...	...	...
b	*b*	*b*
b	*a*	*b*
b	*b*	*b*
...	...	...
b	*b*	*b*
b	*b*	*b*

et

a	*a*	*a*
a	*a*	*a*
...	...	...
a	*a*	*a*
a	*b*	*a*
a	*a*	*a*
...	...	...
a	*a*	*a*
a	*a*	*a*

.

Cette conjecture a été infirmée par deux contre-exemples, l'un montrant que certaines configurations accessibles peuvent comporter un motif d'isolement dans leur ab-Matrice (cf. la Figure 28 (a)), l'autre indiquant qu'il existe des configurations inaccessibles ne présentant aucun de ces motifs dans leur ab-Matrices (cf. la Figure 28 (b)), ce qui permet d'affirmer qu'il est impossible d'affiner cette conjecture pour faire avancer les recherches.

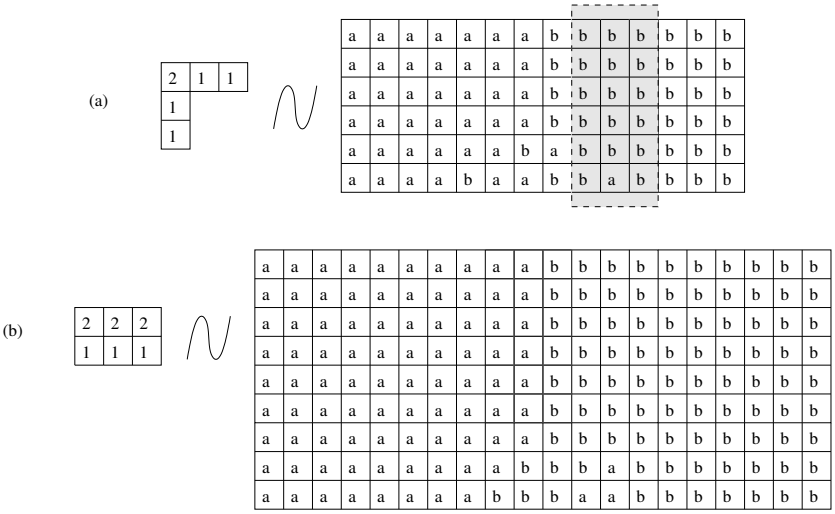


FIG. 28 – Contre-exemples de la conjecture 2.30.

3 Programmation

Pour chacune des approches du problème abordées dans la partie précédente, un programme a dû être développé. Pour les deux programmes réalisés, l'objectif principal était de produire un code tel que le temps de traitement soit raisonnable, ce qui signifie en particulier le plus court possible. Une des conséquences de ce critère a été l'utilisation du langage C largement préférable au Java au niveau de la rapidité en raison de l'utilisation d'un pré-processeur. Par ailleurs, la conception objet n'était pas d'une grande utilité dans ce problème.

Bien entendu, la rapidité d'exécution des programmes ne peut être résolue par la simple utilisation de tel ou tel langage de programmation. En effet, l'algorithmique est le facteur essentiel.

Cette partie présente les deux programmes ayant été réalisés au cours des deux premiers mois de stage. Ce qui peut paraître étonnant, c'est de s'occuper de la pratique avant la théorie mais cela s'explique assez facilement. Il faut comprendre cependant que cette activité de programmation visait soit à améliorer certains logiciels pré-existants mais pas complètement performants, soit à en développer des nouveaux qui, avec les nouvelles versions de ceux pré-existants ont permis de déterminer des conjectures ou de disposer d'exemples qui ont constitué le cadre du stade suivant de l'étude du système.

3.1 Le logiciel probabiliste

3.1.1 Principe général et cahier des charges

Une fois l'étude des probabilités à mettre en jeu réalisée, l'objectif majeur a été de créer un programme permettant d'étudier expérimentalement les points fixes d'un système $BSPM$. Cela revient à simuler un très grand nombre (fini) de chemins de manière à générer automatiquement des statistiques sur les points fixes ayant été atteints. Bien entendu, ce programme devait permettre de choisir pour chaque exécution la distribution de probabilités à utiliser afin de pouvoir comparer les résultats obtenus en fonction des critères choisis.

Les résultats devaient donc être conservés dans des fichiers spécifiques à la nature de l'exécution (c'est-à-dire spécifiques au nombre de grains du système choisi pour la simulation et à la distribution des probabilités choisies), de manière à pouvoir être consultés mais aussi afin de pouvoir y ajouter les résultats de nouvelles exécutions de même nature pour affiner les statistiques.

Ainsi, les différentes statistiques intéressantes lisibles dans un fichier de sauvegarde de résultats peuvent être séparées en deux parties, celles qui sont générales au système étudié (nombre total de chemins simulés, nombre de points fixes trouvés, longueurs minimale et maximale des chemins parcourus) et celles qui donnent des informations sur chacun des points fixes (nombre d'apparitions, pourcentage d'apparition, énergie).

3.1.2 Fonctionnement et temps d'exécution pour l'utilisateur

Ce programme est implémenté de façon à être simple à utiliser du point de vue de l'utilisateur. En effet, en précisant à la suite de la commande le nombre de grains du système ainsi que le nombre de chemins à simuler de manière aléatoire, l'usager accède à un menu lui indiquant de choisir le critère de probabilité désirée parmi les quatre ayant été définies dans la partie 2.2.3. Une fois le choix effectué, la simulation commence et l'utilisateur n'a plus qu'à attendre la terminaison du programme pour visualiser les résultats obtenus. Il est évident que le temps d'exécution varie en fonction des critères que l'utilisateur a choisis (cf. la Figure 29).

	Nombre de grains 10 (energie max = 15) / 100 (energie max = 375)		
	Nombre de chemins		
	1000	10000	100000
Equiprobabilite des grains	0,120s / 2,570s	1,140s / 25,500s	11,180s / 5m14,270s
Probabilite sur les hauteurs de pile	0,110s / 2,520s	1,150s / 25,620s	11,370s / 5m14,540s
Equiprobabilite des piles	0,120s / 6,180s	1,200s / 1m2,550s	11,590s / 10m29,440s
Probabilite sur les differences de hauteurs	0,120s / 5,110s	1,210s / 52,700s	12,060s / 9m16,870s

FIG. 29 – Illustration des temps d'exécution.

On peut remarquer grâce au tableau précédent que le principal critère de variation du temps d'exécution est le nombre de grains du système sur lequel on fait agir le programme. Ceci est bien entendu causé par la profondeur des chemins à simuler. Ici, nous présentons la longueur maximale des chemins simulés dans la première ligne du tableau où nous voyons que cette energie est de 15 pour $BSPM(10)$ et de 375 pour $BSPM(100)$.

Par ailleurs, il est utile de noter que pour un même nombre de grains, le temps d'exécution augmente de façon linéaire. En effet, lorsqu'on multiplie par dix le nombre de chemins, le facteur "temps" est également multiplié par dix.

Enfin, les mises en œuvre de l'équiprobabilité des grains et de la probabilité proportionnelle à la hauteur des piles sont environ deux fois plus rapides (pour un nombre important de grains) que celles de l'equiprobabilité des piles et de la probabilité relative aux différences des hauteurs de piles. En ce qui concerne la différence des hauteurs de piles, cela ne pose pas de problème et se comprend en raison du changement des structures de données et donc des algorithmes. Mais il n'en va pas de même pour l'équiprobabilité des piles. En effet, les structures employées sont identiques à celles des deux premières probabilités énoncées et l'algorithme employé est une copie adaptée de celui utilisé pour

l'equiprobabilité des grains et pourtant, le facteur deux est présent. Plus précisément, lorsqu'on compare les résultats de deux simulations aléatoires de 100000 chemins, l'une illustrant l'équiprobabilité des grains, l'autre l'équiprobabilité des piles, on voit que la deuxième trouve seulement 447 points fixes au lieu de 8494 pour la première, et que les chemins qu'elle "visite" ont une énergie minimale de 260 à la place de 153 pour la première. Il semble donc que l'équiprobabilité des piles parcourt un nombre très inférieur de chemins différents que l'équiprobabilité des grains mais ils sont cependant de manière générale de longueur plus grande, d'où un temps d'exécution plus long.

Toutefois, nous n'avons pas réussi à trouver une explication valable à ce phénomène au niveau algorithmique.

3.1.3 Structures de données

Plusieurs structures de données sont nécessaires pour parvenir à développer ce logiciel. Ces diverses structures devaient respecter la contrainte suivante, à savoir utiliser un minimum d'espace mémoire. Lorsque l'on traite des graphes, il est évident que plus la profondeur augmente, plus le nombre d'éléments à parcourir est important donc plus la taille de l'espace mémoire utilisé augmente. Du coup, si l'on ne gère pas la mémoire de la meilleure manière possible, on arrive vite à sa saturation. L'avantage de ce programme est que le problème relatif à la mémoire n'est pas capital car on ne gère pas l'ensemble du graphe mais un seul chemin.

Ce logiciel ayant pour but de générer aléatoirement des chemins de l'état initial du système jusqu'aux points fixes afin de stocker ces derniers, plusieurs types de données ont été définis.

Tout d'abord, le premier type concerne les sommets du graphe. Chaque sommet du graphe représente une configuration du système. Un sommet est donc caractérisé par un numéro et par une matrice d'entiers correspondant à la partition plane. La structure sous-jacente est donc implémentée en C comme suit :

```
typedef struct struct_sommet sommet;
struct struct_sommet
{
  int num_sommet;   /* Numero du sommet. */
  int **mat_sommet; /* Representation du sommet par une matrice
                       d'entiers. */
};
```

Il faut ensuite trouver un moyen de représenter les sommets correspondants aux points fixes. Cependant, afin d'en tirer des statistiques, un point fixe contient des informations spécifiques comme son énergie et le nombre de fois qu'il apparaît au cours de la simulation. D'où la définition suivante :

```
typedef struct pt_fixe pt_fixe;
struct pt_fixe
{
  int nb_app; /* Frequence d'apparition du point fixe. */
  int nrj;    /* Energie : longueur des chemins y amenant. */
  int **mat;  /* Matrice de representation du point fixe. */
};
```

À ces deux définitions de types, sur lesquelles l'ensemble du programme est basé, sont ajoutées des variables dont les deux principales correspondent à des tableaux de pointeurs, l'un servant à stocker le chemin en cours de simulation et l'autre l'ensemble de tous les points fixes trouvés (cf. la Figure 30). Nous aurions pu utiliser d'autres structures que des simples tableaux mais il s'avère que ces structures se sont avérés suffisament performantes par rapport aux résultats demandés.

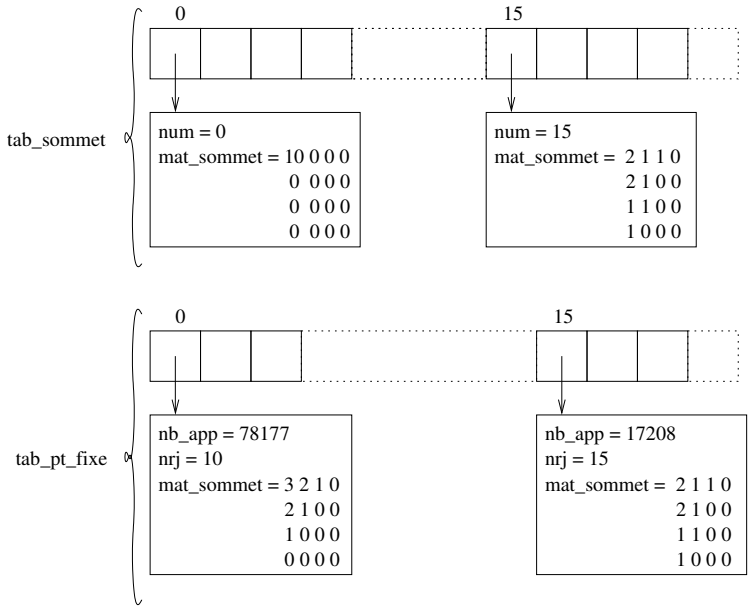


FIG. 30 – Tableau de structures.

De plus, il est intéressant d'observer que la mise en place de la probabilité relative à la différence de hauteurs entre les piles de sable a nécessité l'ajout de matrices spécifiques permettant de modéliser pour chaque pile ses mouvements faisables, les différences de hauteurs qu'elle admet avec ses voisines Sud et Est ainsi que des intervalles de probabilités utiles pour l'utilisation de la fonction *rand()* du C (fonction permettant de générer des nombres aléatoirement).

3.1.4 Principaux algorithmes

Dans cette partie, nous présentons les divers algorithmes mis en jeu afin que le logiciel soit suffisament efficace. Bien entendu, il est fort possible que plus de temps aurait permis

de créer des algorithmes encore plus rapides mais il faut garder en tête que ce programme n'était qu'une petite partie du travail à réaliser. En effet, nous verrons plus tard que le logiciel déterministe a demandé une étude plus conséquente en terme de structures de données et d'algorithmique.

Les parties relatives aux quatre distributions de probabilité sont fondées sur le même algorithme principal. Celle concernant la distribution basée sur la différence des hauteurs de piles est quelque peu différente en raison de la prise en compte des mouvements faisables et non plus des grains ou des piles.

Cependant, ce qui change vraiment est l'implémentation du calcul des probabilités. En effet, selon la distribution choisie, l'éboulement à effectuer n'est pas choisi de la même façon (cf. le Paragraphe 2.2). Il existe par conséquent un algorithme spécifique pour chacun des calculs de probabilités. En réalité, ceci correspond au choix de la pile à faire s'effondrer (et aussi au sens d'éboulement si l'on s'intéresse à la différence des hauteurs).

Nous allons dans un premier temps présenter l'algorithme principal commun pour donner ensuite les particularités dans chacun des différents cas.

- **Algorithme principal**

  Le fonctionnement général du programme est le suivant. Lorsque l'utilisateur lance le programme en utilisant la commande

  ./simul <nombre de grains> <nombre de chemins>

  ce dernier crée la configuration initiale du système pour engendrer au fur et à mesure de l'exécution l'ensemble des sommets du graphe qui compose le chemin suivi. La procédure est la même pour chaque sommet. Le logiciel analyse la configuration courante, détermine d'abord si au moins un éboulement est possible (deux mouvements sont toujours possibles à partir de la configuration initiale). Si aucun mouvement n'est possible, la configuration en question est un point fixe et la simulation peut s'attacher à un nouveau chemin. Sinon, si plusieurs piles peuvent s'effondrer, on en choisit une au hasard en fonction de la distribution de probabilité choisie par l'utilisateur et on effectue au hasard l'un des mouvements possibles (ceci n'est pas fait dans le cas de la probabilité relative à la différence des hauteurs de piles car celle-ci choisit directement le mouvement et non pas la pile). On obtient alors une nouvelle configuration (fille de la configuration initiale). C'est alors ce nouveau sommet du graphe que le programme va étudier. Il effectue alors les mêmes opérations jusqu'à obtenir un point fixe que l'on conserve dans le tableau prévu à cet effet.

  Une fois le point fixe obtenu, le programme se relance pour engendrer un nouveau chemin et s'arrête une fois que tous les chemins demandés soient simulés. À ce moment là, le programme effectue un parcours du tableau de points fixes afin de les enregistrer avec les statistiques dans un fichier de sauvegarde propre à la nature de l'exécution.

D'où l'algorithme (simplifié) ALGO PRINCIPAL.

Algorithme 1 : ALGO PRINCIPAL

tant que tous les chemins demandés ne sont pas simulés **faire**
Initialisation du tableau de sommets du chemin :
> Création du sommet initial;
> Insertion du sommet initial dans le tableau de sommets;

Initialisation d'un générateur aléatoire de nombres;

Complétion récursive du tableau de sommets du chemin à partir du sommet précédemment créé :
> Réallocation du tableau de sommet si nécessaire;
> **si** le sommet précédemment créé est un point fixe **alors**

. Marquage de la terminaison du chemin dans le tableau de sommets;
. Enregistrement du point fixe dans le tableau de points fixes;
. Arrêt de la complétion du chemin et changement d'itération de la boucle principale pour la simulation d'un nouveau chemin;

finsi
> Recherche aléatoire du mouvement à effectuer (partie detaillée dans le paragraphe suivant en fonction des probabilités mises en jeu);
> Création d'un nouveau sommet en fonction de l'éboulement choisi;
> Insertion du nouveau sommet dans le tableau de sommets;
> Relancement de la complétion récursive sur le nouveau sommet;

Libération de l'espace mémoire occupé par le tableau contenant les différents sommets du chemin simulé;
fintq
Enregistrement de l'ensemble des points fixes dans un fichier;

- **Recherche aléatoire du mouvement** (se référer au Paragraphe 2.2.3)

  Soit n le nombre de grains du système.

  ○ Équiprobabilité des grains

  Chaque grain est numéroté avec les nombres de 1 à n dans le sens des lignes de la matrice (c'est-à-dire que si la pile $(1,1)$ contient k grains et la pile $1,2$ en contient $k-5$, la pile $(1,1)$ sera constituée des grains numérotés de 1 à k et la pile $(1,2)$ des grains numérotés de $k+1$ à $2k-5$, et ainsi de suite en procédant ligne par ligne). Chaque grain a $\frac{1}{n}$ chance d'être celui choisi pour tomber. Un entier entre 1 et n est alors tiré aléatoirement. Comme seuls les grains au sommet des piles peuvent bouger, il est nécessaire de relancer la procédure de choix jusqu'à ce qu'un grain situé en haut d'une pile soit sélectionné. Une fois ce grain trouvé, il faut déterminer dans quelle(s) direction(s) il peut aller. Si plusieurs sont possibles, il faut en sélectionner une au hasard en tirant aléatoirement un nombre entre 1 et 2. Ainsi, si on obtient 1, on choisit arbitrairement la direction Sud et sinon, on choisit la direction Est.

  D'où l'algorithme ALGO ÉQUIPROBA_GRAINS.

  ○ Équiprobabilité des piles

  Chaque pile potentiellement apte à subir un éboulement est numérotée avec les nombres de 1 à $\frac{\tau(\tau-1)}{2}$ ligne par ligne, avec τ la taille des partitions planes du système. Ainsi, la première ligne de la matrice contient les piles numérotées de 1 à

Algorithme 2 : ALGO ÉQUIPROBA_GRAINS

```
Données : Entier num_grain, dir;
          Entier[2] coord_pile;
          /* Direction valant "non", "sud", "est" ou "sud et est" */
          Chaîne[10] dir_poss;
début
    /* Soit tm la taille des partitions planes du système */

    num_grain := Génération d'un nombre aléatoire entre 1 et n;
    coord_pile := Recherche des coordonnées de la pile a ébouler :
    > Données : Entier somme_tmp, i, j;
    > somme_tmp := 0;
    > /* Parcours des piles de la configuration C ligne par ligne */
    > pour i de 1 à tm faire
        pour j de 1 à (tm-i) faire
            somme_tmp := somme_tmp + C[i][j];
            /* Si le grain choisi est en haut de la pile courante */
            si somme_tmp = num_grain alors
                coord_pile[1] := i;
                coord_pile[2] := j;
                retourner coord_pile;
            finsi
            /* Sinon si le grain est dans la pile courante mais pas au sommet */
            si somme_tmp > num_grain alors
                coord_pile := NULL;
                retourner coord_pile;
            finsi
        finpour
      finpour

    //** La partie suivante est commune à l'équiprobabilité des grains
    //** celle des piles et à la probabilité relative à la hauteur de pile.
    //** Elle ne sera donc pas réécrite dans les deux algorithmes suivants.

    si coord_pile = NULL alors
        Libération de l'espace mémoire occupé par coord_pile;
        On relance la procédure de recherche du mouvement ;
    finsi
    sinon
        dir_poss := Recherche des directions d'éboulement possibles;
        /* Cas ou l'eboulement est impossible */
        si dir_poss = "non" alors
            Libération de l'espace mémoire occupé par coord_pile;
            Libération de l'espace mémoire occupé par dir_poss;
            On relance la procédure de recherche du mouvement;
        finsi
        /* Si l'éboulement est possible dans les deux directions, on en choisit une au hasard */
        si dir_poss = "sud et est" alors
            dir := Génération d'un nombre aléatoire entre 1 et 2;
            si dir = 1 alors  dir_poss := "sud";
            sinon  dir_poss := "est";
        finsi
    finsi
fin
```

$\tau - 1$, la deuxième les piles de τ à $\tau + \tau - 2$, et ainsi de suite jusqu'à la dernière pile ne contenant que la dernière pile numérotée $\frac{\tau(\tau-1)}{2}$. Chaque pile a donc $\frac{2}{\tau(\tau-1)}$ chances d'être celle sélectionnée pour l'avalanche. Il faut ensuite savoir si la pile choisie respecte les conditions de transition. Si tel n'est pas le cas, il faut en choisir une autre de la même manière.

D'où l'algorithme ALGO ÉQUIPROBA_PILES.

Algorithme 3 : ALGO ÉQUIPROBA_PILES

```
Données : Entier num_pile, dir;
          Entier[2] coord_pile;
          /* Direction valant "non", "sud", "est" ou "sud et est" */
          Chaîne[10] dir_poss;
début
    /* Soit tm la taille des partitions planes du système */

    num_pile := Génération d'un nombre aléatoire entre 1 et tm(tm+1)/2 - tm;
    coord_pile := Recherche des coordonnées de la pile a ébouler :
    > Données : Entier somme_tmp, i, j;
    > somme_tmp := 0;
    > /* Parcours des piles de la configuration C ligne par ligne */
    > pour i de 1 à (tm-1) faire
        pour j de 1 à (tm-i-1) faire
            somme_tmp := somme_tmp + 1;
            /* Si la pile choisie est la pile courante */
            si somme_tmp = num_pile alors
                /* Si la pile n'est pas assez haute */
                si C[i][j] < 2 alors
                    coord_pile := NULL;
                    Renvoie coord_pile;
                finsi
                sinon
                    coord_pile[1] := i;
                    coord_pile[2] := j;
                    retourner coord_pile;
                finsi
            finsi
        finpour
      finpour

    (La suite de l'algorithme est identique à celui de l'équiprobabilité des grains.)
fin
```

○ Probabilité proportionnelle à la hauteur des piles

Ici, ce ne sont pas les piles qui sont numérotée mais les grains, de façon analogue à la numérotation entreprise pour l'équiprobabilité des grains. Si l'on considère n_i le nombre de grains contenu dans la i-ième pile, la probabilité qu'une pile soit selectionnée pour donner un grain à l'une de ses voisines Sud ou Est est de $\frac{n_i}{n}$. Comme pour les probabilités précédentes, ce n'est pas parce que la pile est choisie qu'un mouvement est possible. Une vérification supplémentaire est par conséquent nécesssaire.

D'où l'algorithme ALGO PROBA_PILES_HAUTES.

○ Probabilité relative aux différences de hauteurs

Algorithme 4 : ALGO PROBA_PILES_HAUTES

```
Données : Entier num_grain, dir;
          Entier[2] coord_pile;
          /* Direction valant "non", "sud", "est" ou "sud et est" */
          Chaîne[10] dir_poss;
début
    /* Soit tm la taille des partitions planes du système */

    num_grain := Génération d'un nombre aléatoire entre 1 et n;
    coord_pile := Recherche des coordonnées de la pile a ébouler :
    > Données : Entier somme_tmp, i, j;
    > somme_tmp := 0;
    > /* Parcours des piles de la configuration C ligne par ligne */
    > pour i de 1 à tm faire
        pour j de 1 à (tm-i) faire
            somme_tmp := somme_tmp + C[i][j];
            /* Si le grain choisi est dans la pile courante */
            si somme_tmp ≥ num_grain alors
                coord_pile[1] := i;
                coord_pile[2] := j;
                retourner coord_pile;
            finsi
        finpour
      finpour

    (La suite de l'algorithme est identique à celui de l'équiprobabilité des grains.)
fin
```

Ici, il est indispensable de procéder de manière différente. En effet, à la différence des autres probabilités, lorsqu'un mouvement est choisi pour être effectué, ce dernier est réalisable. On ne repasse donc pas par une nouvelle vérification de faisabilité car le mouvement est déjà déterminé. Ceci présente bien évidemment des difficultés algorithmiques pour la mise à jour des mouvements faisables. À ce sujet, nous avons adopté une mise à jour des différences de hauteurs de piles en $\mathcal{O}(1)$. En effet, si l'on prend comme exemple une transition Est, on connait le nombre de différences de hauteur qui vont changer. Elles se situent bien sûr toujours à la même position. (cf la Figure 31). Cependant, la complexité en temps est tout de même supérieure à celle des algorithmes précédents en raison du parcours de la matrice courante pour séterminer le mouvement faisable à effetuer (un parcours identique est effectué pour les autres probabilités) et de la mise à jour permanente des intervalles de probabilités associés à chaque mouvement faisable (parcours équivalent supplémentaire). La complexité, même si elle reste du même ordre admet néanmoins un facteur 2.

En conséquence, à chaque étape, la somme de toutes les différences de hauteurs supérieures à deux (valant $2 \times n$ pour la configuration initiale) est recalculée. De plus, différentes matrices sont remises à jour :

– la matrice des mouvements faisables : il s'agit d'une matrice d'entiers de même taille que les partitions planes du système. Ainsi, les éléments de cette matrice prennent pour valeur -1 si aucun mouvement n'est possible à partir de la pile de même coordonnées, 0 si seul le mouvement vers le Sud est envisageable,

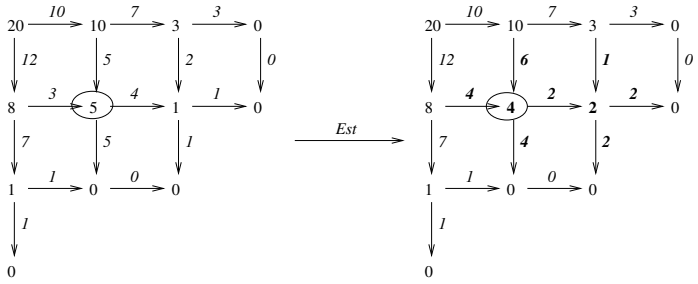


FIG. 31 – Mise à jour des différences de hauteur en $\mathcal{O}(1)$.

1 pour un mouvement vers l'Est et 2 pour signaler que les deux mouvements sont faisables,

– les deux matrices contenant, pour chaque pile, l'une la différence de hauteur qu'elle admet avec sa voisine du Sud et l'autre la différence de hauteur qu'elle admet avec sa voisine Est,

– les deux matrices "d'intervalles de probabilités", l'une pour la direction Sud, l'autre pour la direction Est : il s'agit de deux matrices de tableaux de deux entiers (int ***). Le nombre entier représentatif de la probabilité (entre 1 et le total des différences de hauteur) est compris dans un intervalle (déterminant par conséquent la direction de l'éboulement à effectuer) donné par un tableau dont les coordonnées dans la matrice correspondent aux coordonnées de la pile à faire s'effondrer.

La Figure 32 illustre le mode d'utilisation de ces matrices.

D'où l'algorithme ALGO PROBA_DIFF_HAUTEUR.

Algorithme 5 : ALGO PROBA_DIFF_HAUTEUR

```
Données : Entier proba, dir;
          /* coord_pile[3] indique le mouvement à effectuer : 0 pour le Sud et 1 pour l'Est */
          Entier[3] coord_pile;
          /* Direction valant "non", "sud", "est" ou "sud et est" */
          Chaîne[10] dir_poss;
début
    /* Soit tm la taille des partitions planes du système. Considérons par ailleurs la somme des différences de hauteurs (somme_diff_hauteur) ainsi que toutes les matrices citées précédemment (mat_mvt_faisable, mat_diff_sud, mat_diff_est, mat_proba_sud, mat_proba_est) comme des variables globales initialisées au début du programme et mises à jour à chaque étape */

    proba := Génération d'un nombre aléatoire entre 1 et somme_diff_hauteur;
    coord_pile := Recherche des coordonnées de la pile à ébouler :
    > Données : Entier i, j;
    > /* Parcours des piles de la configuration C ligne par ligne */
    > pour i de 1 à tm faire
        pour j de 1 à tm faire
            /* Si la probabilitécalculée est incluse dans l'intervalle des probabilités Sud de la pile courante */
            si (proba ≥ mat_proba_sud[i][j][1]) et (proba ≤ mat_proba_sud[i][j][2]) alors
                coord_pile[1] := i;
                coord_pile[2] := j;
                coord_pile[3] := 0;
                Renvoie coord_pile;
            finsi
            /* Si la probabilitécalculée est incluse dans l'intervalle des probabilités Est de la pile courante */
            si (proba ≥ mat_proba_est[i][j][1]) et (proba ≤ mat_proba_est[i][j][2]) alors
                coord_pile[1] := i;
                coord_pile[2] := j;
                coord_pile[3] := 1;
                Renvoie coord_pile;
            finsi
        finpour
      finpour
    si coord_pile[3] = 0 alors
        Création directe de la nouvelle configuration en effectuant une transition Sud ;
    finsi
    sinon
        Création directe de la nouvelle configuration en effectuant une transition Est ;
    finsi
fin
```

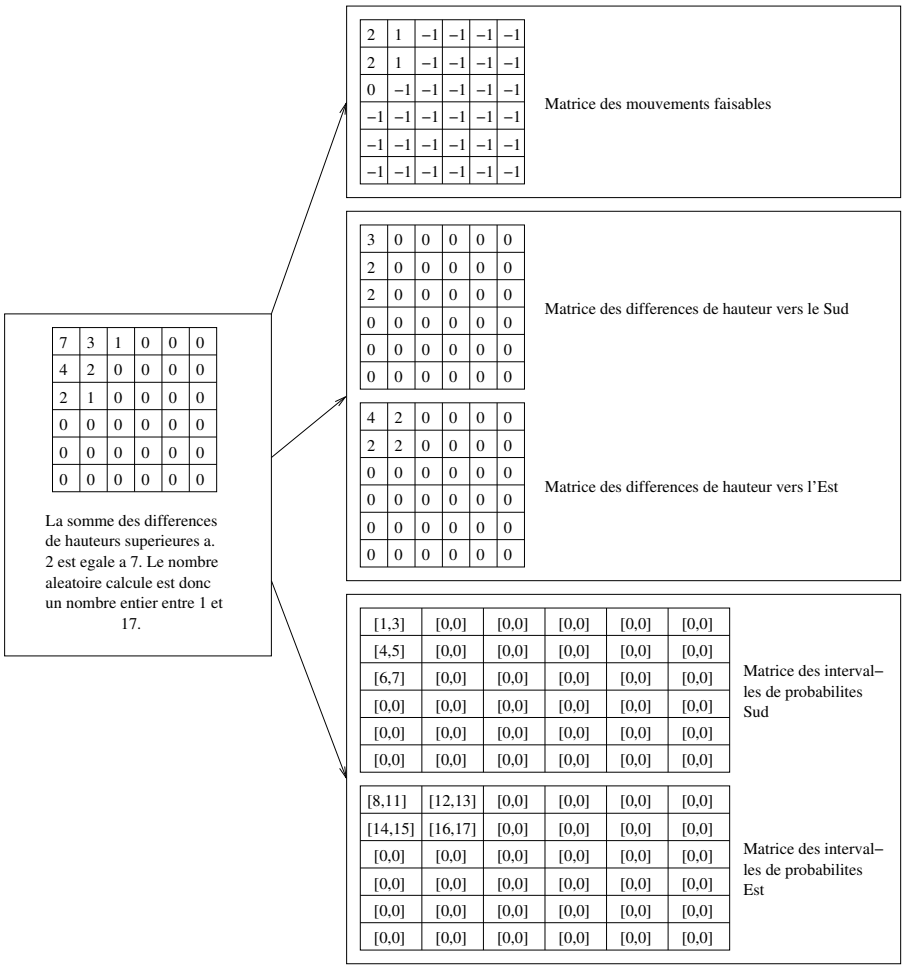


FIG. 32 – Les matrices utiles pour les différences de hauteurs.

3.2 Le logiciel déterministe

Maintenant que nous avons expliqué la phase du stage concernant la conception et le développement du logiciel probabiliste, il est temps de nous intéresser plus en détail au logiciel déterministe. L'implémentation de ce programme était le sujet de TER (Travail d'Études et de Recherches) de quatre étudiants de maîtrise l'année dernière. Cependant, Roberto Mantaci m'a expliqué avant de commencer mon stage que le logiciel réalisé au cours du TER comportaient des faiblesses algorithmiques et qu'une bonne chose à faire était soit de le reprendre pour l'améliorer soit de le réimplanter complètement.

Après avoir établi le principe général de ce logiciel déterministe, nous donnerons un rapide aperçu du fonctionnement du programme pré-existant afin de présenter et d'expliciter le travail effectué pour en créer une nouvelle version.

3.2.1 Principe et problématique

Le principe en lui-même de ce logiciel est assez simple. On sait que le modèle mathématique sous-jacent de tout système $BSPM$ est un graphe fini comportant plusieurs points fixes. De plus, la taille de ce graphe est proportionnelle au nombre de grains du système. Elle varie en effet de façon exponentielle (cf. la Figure 33).

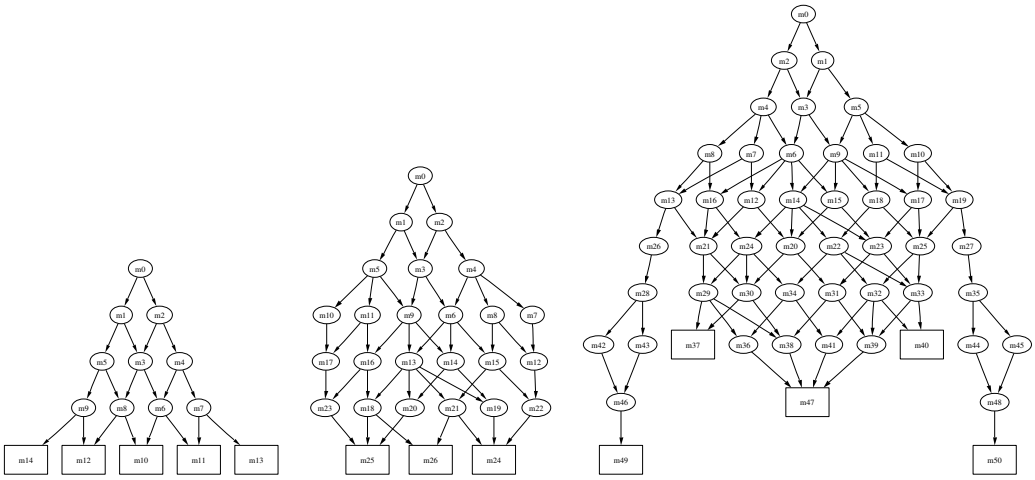


FIG. 33 – Taille des graphes : $BSPM(5)$, $BSPM(6)$, $BSPM(7)$.

L'objectif est donc de réussir à engendrer le graphe représentatif de tout système $BSPM$ à partir d'un nombre de grains quelconque donné par l'utilisateur.

Lorsqu'on souhaite traiter des graphes dans leur totalité en informatique, plusieurs problèmes se posent, que ce soit au niveau de la rapidité des programmes ou au niveau de l'espace mémoire qu'ils utilisent. Ici, ces deux problèmes n'ont pu être évités. Une étude approfondie a donc été indispensable pour minimiser cette complexité en temps et en espace.

3.2.2 Le logiciel pré-existant

En entamant cette partie du projet, on m'a expliqué que la version du logiciel existant que le LIAFA possédait ne fonctionnait pas. En réalité, une version *alpha* avait été réalisée au cours du TER mais s'était avérée trop lente. Du coup, il avait été demandé aux étudiants de l'améliorer et d'en créer une version *beta*. Les étudiants ont donc renvoyé une nouvelle version. Durant la phase de test de cette dernière version, les chercheurs se sont rendus compte qu'elle ne fonctionnait pas et que, malheureusement, la première version n'avait pas été conservée.

Par conséquent, avant de savoir si la meilleure chose à faire était de le reprendre comme base ou de le réimplanter à partir de nouvelles fondations, le premier objectif que je me suis fixé a été d'étudier le code existant afin d'en tirer les avantages et les inconvénients. Au cours de cette étape, les lacunes algorithmiques recensées se sont montrées importantes. En effet, la génération et le parcours du graphe était abordés avec des listes chaînées, ce qui est un moyen très lent pour ce genre de traitement.

Ainsi, voyant que les structures de données n'étaient pas les plus appropriées, nous avons décidé de reprendre le programme à sa source pour lui offrir de nouvelles bases.

3.2.3 Le nouveau logiciel

Contrairement au programme existant, nous ne nous sommes intéressés cette année qu'à l'algorithmique et en aucun cas nous nous sommes fixés d'adapter un environnement graphique à ce nouveau logiciel. Cela aurait à coup sûr fait perdre un temps précieux utiles pour les recherches à effectuer.

3.2.3.1 Utilisation

Le logiciel nommé "tas3d" nouvellement développé l'a été de façon à être facile à utiliser. Ainsi, quand le progamme est lancé, l'usager n'a qu'à attendre sa terminaison pour visualiser le bilan obtenu, présent dans le répertoire "résultats" sous la forme de fichiers .txt. Ces fichiers de résultats présentent les configurations créées, énergie par énergie.

Remarque 3.1 *Il est utile de noter que plusieurs fichiers peuvent être engendrés pour un même nombre de grains. Cela provient de l'option choisie. Nous y reviendrons lors de la présentation des diverses options.*

De plus, ce logiciel offre différentes options qui étaient incluses dans le cahier des charges. Pour être prises en compte, elles doivent être passées à la commande suivante :

./tas3d [-abf] [-dot] [-pf] <nombre de grains (nbg)>

Sans option, le programme conserve dans un fichier de nom "res_<nbg>_all.txt" l'ensemble de toutes les configurations du système.

L'option -dot permet de conserver dans un fichier de nom "res_<nbg>_graph.dot" le graphe du modèle étudié sous forme de liste d'arêtes. Ce ficheir est lisible par le logiciel *"Dotty"* qui effectue l'implémentation graphique.

L'option -pf évite de conserver l'ensemble de toutes les configurations. Elle permet d'enregistrer uniquement les points fixes dans le fichier de nom "res_<nbg>_pf.txt.

Enfin, l'option -abf donne la possibilité à l'utilisateur d'enregistrer, en plus des matrices représentatives des configurations, les 2 modèles alternatifs du système que sont les ab-Matrices et les f-Matrices.

Remarque 3.2 *Ces trois options peuvent être combinées dans n'importe quel ordre selon la volonté de l'utilisateur. Il pourra donc choisir (ceci n'est qu'un exemple) de n'afficher que les points fixes avec leurs modèles alternatifs en tapant "./tas3d -pf -abf 15".*

3.2.3.2 Idées générales et structures de données

La création du graphe du *BSPM* demandé est basé sur un parcours en largeur. Sachant qu'une même profondeur du graphe correspond à une même génération de configurations, il a semblé assez logique d'utiliser ce type de parcours. Ainsi, le graphe est engendré génération par génération en utilisant un parcours en largeur.

Néanmoins, la nature du parcours n'est pas un facteur déterminant au niveau de la rapidité du logiciel. C'est pourquoi nous allons à présent nous attacher à expliquer les différentes idées de structures de données auxquelles nous avons pensées par ordre chronologique.

Il faut savoir que l'objectif majeur auquel il fallait bien penser était la complexité en temps. En effet, bien que la complexité en mémoire soit aussi un facteur important, elle reste de second plan. Le but était donc de trouver des structures de données telles que la rapidité d'exécution soit maximale et d'adapter ensuite ces structures de manière à minimiser l'espace mémoire.

La première chose à laquelle il a fallu penser est la façon de représenter les configurations. Pour ce faire, il a fallu déterminer le minimum d'informations indispensables les concernant. Ceci est apparu assez intuitif et a donc été assez rapide à concevoir. Afin de distinguer rapidement chacune des configurations, elles ont toutes un numéro qui leur est propre. Par ailleurs, il est intéressant de conserver des tableaux contenant les liens de parenté entre les configurations ainsi que la taille de ces derniers. Enfin, en raison du fait qu'un sommet du graphe soit principalement défini par sa partition plane correspondante, il a été indispensable de stocker sa matrice d'entiers représentative. Les configurations sont donc définies de la manière suivante :

```
typedef struct configuration configuration;
struct configuration
{
  int num;       /* Numero de la configuration. */
  int *meres;    /* Tableau contenant les numeros des configurations meres,
                    le numero d'une configuration etant son indice dans le
                    tableau des configurations. */
  int *filles;   /* Tableau contenant les numeros des configurations
                    filles. */
  int nb_meres;  /* Nombre de configurations meres. */
  int nb_filles; /* Nombre de configurations filles. */
  int **mat;     /* Matrice d'entiers pour la representation "physique" de
                    la configuration. */
};

configuration **tab_config_gen_paire;
configuration **tab_config_gen_impaire;
```

Une fois le choix de la structure des configurations définie, il a fallu s'attacher à l'étape délicate, à savoir la manière de les stocker. Nous avons tout d'abord mis de côté l'emploi des listes chaînées car elles possèdent l'inconvénient de ne pas permettre l'accès direct à ses éléments, ce qui ne pouvait pas nous convenir si nous voulions obtenir une complexité en temps minimale. Le but était alors d'accéder aux configurations en se rapprochant au maximum d'une compléxité en $\mathcal{O}(n)$ tout en utilisant un minimum d'espace mémoire.

La première idée a été de rajouter à la structure des configurations un champ entier renseignant sur son énergie et de gérer les accès aux configurations par l'intermédiaire de tables de hachage dynamique (cf. la Figure 34).

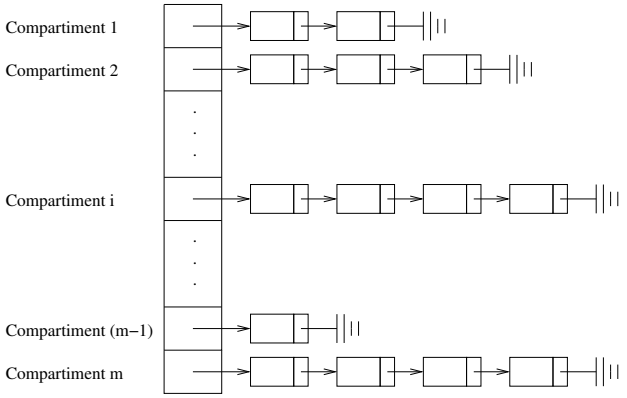


FIG. 34 – Table de hachage dynamique.

L'objectif de ces tables de hachage est séparer les éléments à stocker dans des compartiments, d'affecter une "clé" unique à chaque compartiment calculable une fonction de hachage. Ainsi, en calculant la clé pour une configuration, on détermine dans quel compartiment elle est et on n'a plus qu'à la rechercher dans ce compartiment. Par conséquent, plus le compartiment est petit, plus l'accès est rapide.

Évidemment, cette technique aurait permis d'obtenir un programme rapide mais n'aurait pas donné la possibilité d'avoir un accés direct à une configuration pour connaître ses filles (par exemple). Le cas permettant ceci est d'avoir autant (ou plus) de compartiments que de configurations. Mais c'est très compliqué de trouver une fonction de hachage de la sorte qui soit, en plus, calculable rapidement. En admettant que les compartiments de la table de hachage contiennent m éléments, la complexité de cette solution pour accéder à une configuration est donc déterminée, dans le pire des cas, par un parcours des m éléments du compartiment la contenant, pour lesquels chaque matrice de représentation est parcourue. D'où une complexité de $\mathcal{O}(n \times m)$.

Souhaitant obtenir une complexité en temps de $\mathcal{O}(n)$ dans le pire des cas, il a fallu trouver un autre moyen de stocker ces configurations. Nous avons alors pensé à utiliser un arbre lexicographique pour stocker chaque génération. C'est un moyen très efficace pour stocker les matrices des configurations à la manière d'un dictionnaire pour en permettre un accès rapide.

Pour comprendre le principe, il faut imaginer un arbre dans lequel chaque branche est étiquetée par un nombre qui correspond à un coefficient de matrice. L'arbre donne une représentation des matrices correspondante à une lecture de celle-ci ligne par ligne de la gauche vers la droite et du haut vers le bas. Seuls les coefficients strictement supérieurs à 0 sont donnés par l'arbre afin de traiter une profondeur d'arbre minimale. Chaque ligne est séparée de sa suivante par une branche étiquetée par 0. Par exemple, la configuration

$$\alpha = \begin{matrix} 4 & 2 & 1 \\ 3 & & \\ 1 & & \end{matrix}$$

sera représentée par le mot $4\,2\,1\,0\,3\,0\,1$ et ce mot sera l'étiquette du chemin de l'arbre lexicographique atteint de la racine à la feuille qui correspond à α. L'avantage de ce concept est que l'on peut stocker dans le même arbre une "infinité" de matrices différentes en utilisant un minimum d'espace. En effet, deux matrices commençant par les mêmes coefficients seront définies dans l'abre lexicograhique par des premières branches identiques comme le montre la Figure 35.

Pour rendre encore meilleure cette structure définie plus loin, les feuilles de l'arbre contiennent le numéro de la configuration correspondante, ce qui permet un accés direct aux renseignements de cette dernière (stockée dans un tableau). À ce sujet, il existe deux tableaux de configurations, l'un pour les configurations de génération paire et l'autre pour les configurations de génération impaire et deux arbres lexicographiques pour les mêmes raisons. Ceci nous permet de ne conserver que deux générations au même instant et donc réduire au maximum l'espace mémoire utilisé.

Par l'utilisation de ces arbres, l'accès à une configuration a une complexité en temps égale à $\mathcal{O}(n)$. Ceci permet également de parcourir toute une génération, c'est-à-dire tout l'arbre lexicographique, en un temps de l'ordre de $\mathcal{O}(n \times N)$ où N est le nombre de

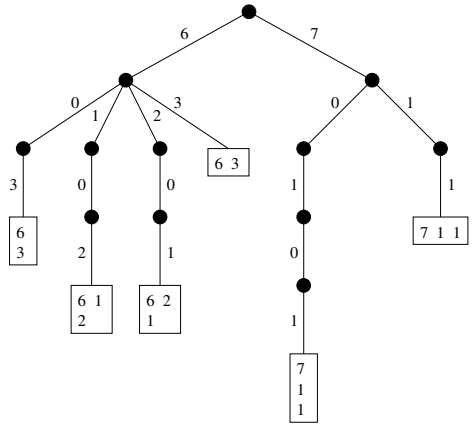


FIG. 35 – Arbre lexicographique des configurations d'énergie 3 de $BSPM(9)$.

configuration representées par l'arbre (son nombre de feuilles). Ici, on peut se demander si la simple utilisation du tableau de configuration ne permettrait pas un parcours d'une génération aussi rapide. Là aussi, il faudrait en effet parcourir le tableau contenant N configuration, chacune de ces N étapes nécessitant un parcours d'une matrice, d'où à première vue une complexité en $\mathcal{O}(n \times N)$. Toutefois, dans le cas du tableau, la complexité est déterminable de façon certaine. On note donc $\Theta(n \times N)$.

Remarque 3.3 *Il faut savoir que la différence entre ces deux procédés n'est pas quantifiable avec exactitude. Le parcours infixe (qui revient à un parcours en profondeur) d'une génération codée sous la forme de tableau revient en fait exactement à celui codée avec le pire arbre possible, à savoir un arbre ne possédant que des branches différant directement au niveau de la racine, ce qui ne se vérifie que dans des cas extrèmement rares et pour des toutes petites valeurs de n ($n \leq 4$).*

Par conséquent, l'utilisation de l'arbre lexicographique est préférable. La structure utilisée est donc définie en C de la manière suivante :

```
typedef struct generation generation;
struct generation
{
  int num_config;
  generation **tab;
};

generation *gen_paire;
generation *gen_impaire;
```

3.2.3.3 Principaux algorithmes

En dehors de la partie théorique, c'est la conception et l'implémentation de ce logiciel qui m'a posé le plus de difficultés. Les diverses subtilités entraînées ont fait que toute cette deuxième étape de la partie pratique du stage a passé très rapidement en raison de la réflexion en découlant.

Bien sûr, les problèmes liés à l'implémentation sont assez classiques pour un informaticien et ne sont pas de premier ordre. C'est pourquoi nous aborderons ici les différents algorithmes mis en oeuvre pour le logiciel "tas3d" sans évoquer le code résultant dans le langage C.

En partant de la génération paire 0 facile à initialiser parce qu'elle ne contient que la configuration initiale, on crée la génération impaire 1. Ensuite, il faut procéder de la même façon pour créer la génération 2 à partir de la génération 1 et ainsi de suite jusqu'à obtenir dernière génération, c'est-à-dire celle qui ne contient qu'une ou plusieurs configurations qui sont arrivées à stabilité, à savoir des points fixes. Cela amène à la conception d'un premier algorithme de création de la génération suivante d'une génération connue. Le fonctionnement de cette partie (essentielle) du logiciel est le suivant :

- On parcourt toutes les configurations de la génération courante. Ce parcours est un parcours infixe (qui revient à un parcours en profondeur). Lorsqu'on tombe sur un noeud auquel est affecté un numéro différent de -1, on sait que le chemin suivi dans l'arbre pour y arriver représente une configuration.
- On crée l'ensemble des configurations filles de cette dernière.
- On insère si nécessaire ces configurations filles dans la nouvelle génération. Cette étape de recherche et d'insertion d'une configuration est celle qui sera détaillée dans ce rapport. Elle se décompose en deux fonctions présentées dans les algorithmes ALGO INSERTION_CONFIGURATION appelé par ALGO INSERTION_CONFIGURATION_SI_NÉCESSAIRE.

Une fois la nouvelle génération créée, afin de ne conserver que deux générations au maximum à un instant donné, il est indispensable de parcourir l'ancienne génération pour la sauvegarder dans le fichier résultat approprié en fonction de la commande d'exécution passée, puis de la supprimer et ainsi de libérer tout l'espace mémoire qu'elle occupe. Ceci implique un autre algorithme que nous pourrions appeler algorithme de sauvegarde. Cependant, ce dernier étant fondé sur le même parcours en profondeur de l'arbre de génération.

Ensuite, au cours des travaux de recherches, lorsque nous avons analysé les modèles alternatifs des tas de sable, nous nous rendu compte qu'il serait grandement bénéfique de ne plus les créer à la main et de pouvoir les engendrer automatiquement par le biais d'un petit programme. Ainsi, un petit logiciel "indépendant" capable, à partir d'une matrice donnée de générer automatiquement son ab-Matrice et sa f-Matrice correspondante a été implémenté. Ce programme est basé sur l'algorithme ALGO AB_F_MATRICE qui a par la suite été intégré au logiciel "tas3d".

Nous allons par conséquent présenter ci-dessous ces trois algorithmes qui représentent

les algorithmes les plus importants du logiciel déterministe.

Remarque 3.4 *Les trois algorithmes sont basés sur les structures des configurations et des générations qui ont été définies précédemment en C. Ainsi, nous utiliserons des algorithmes ayant une syntaxe assez proche du C.*

3.2.3.4 Limites

L'ensemble des algorithmes mis en oeuvre pour réaliser ce logiciel déterministe ont permis d'obtenir des résultats tr'es satisfaisants en terme de rapidité. En effet, le graphe du système $BSPM(35)$ qui comporte 5654124 sommets répartis en 105 générations est engendré en treize minutes et cinquante-cinq secondes.

Néanmoins, il n'en demeure pas moins que le traitement de graphes à croissance exponentielle tels que ceux permettant de représenter un système $BSPM(n)$ n'ecessite un espace mémoire important. De ce fait, même si l'efficacité du logiciel est incontestable, son utilisation s'avère limitée par les capacités mémoire des machines sur lesquelles il est exécuté. C'est la raison pour laquelle nous avons fait en sorte d'utiliser uniquement l'espace m'emoire nécessaire, en ne conservant au maximum que deux générations.

La solution préconisée au départ, qui aurait fait gagner de façon non négligeable de l'espace mémoire, aurait été de conserver de la même manière ces deux générations mais en supprimant à chaque étape, lors de la création d'une nouvelle génération, les configurations traitées de l'ancienne génération au fur et à mesure. Mais il fallait conserver un certain ordre de création, à savoir traiter les configurations de la génération mère dans l'ordre où elles ont été engendrées. Cela revient en fait à supprimer le premier élément du tableau des configurations mères après que les filles de la configuration correspondante ont été engendrées.

Durant la phase de réalisation de ce logiciel, il a fallu en permanence trouver des compromis entre la complexité en temps et la complexité en mémoire. Du coup, l'idée pour résoudre ce problème était d'arriver à désallouer le premier élément du tableau en conservant les suivants sans en faire de recopie dans un nouveau tableau (ce qui n'était pas un compromis acceptable pour la complexité en temps). Or ceci est impossible en C. En effet, on ne peut désallouer un emplacement mémoire par la fonction *"free()"* que lorsque ce dernier a été alloué par un appel à la fonction *"malloc()"*. Or, l'allocation des tableaux est effectuée sur le premier élément du tableau (au moment de l'allocation). Donc cette solution, qui semblait la seule parfaitement adaptée, n'a pu être mise en place.

En conséquence, l'exécution qui implique la création de deux tableaux de configurations trop importants (il s'agit de l'exécution du logiciel pour créer $BSPM(38)$) amène à un manque d'espace mémoire et se termine brusquement en indiquant que la taille de l'espace mémoire disponible est insuffisante. Ceci intervient sur la machine "calcul" du LIAFA qui dispose d'un giga-octets de mémoire centrale.

Algorithme 6 : ALGO INSERTION_CONFIGURATION

```
/* Fonction recherchant une configuration dans une génération selon les cas et l'insérant si elle n'a pas été trouvée.
g : la génération dans laquelle la recherche est effectuée.
c : la configuration à rechercher.
nv_gen : 0 dans le cas de la création d'une nouvelle génération, 1 sinon.
Renvoie -1 si la configuration n'a pas été trouvée et le numéro de la configuration dans le cas contraire. */
Fonction inserer_config(generation *g, configuration *c, Entier nv_gen) : Entier;
Données : /* Compteurs pour le parcours de la matrice. */
          Entier i, j;
          /* Génération temporaire pour le parcours de l'arbre de générations. */
          generation *g_tmp;
début
    g_tmp := g;
    pour i de 1 à τ faire
        pour j de 1 à τ faire
            /* Si la valeur de l'élément courant de la matrice est 0, soit il s'agit d'un changement de ligne, soit il
            s'agit de la fin de la matrice. */
            si c->mat[i][j] = 0 alors
                /* S'il s'agit de la dernière ligne "affectée" de la matrice. */
                si (i = (τ-1)) ou (c->mat[i+1][0] = 0) alors
                    /* Cas où la configuration est déjà présente. */
                    si g_tmp->num_config ≠ -1 alors
                        retourner g_tmp->num_config;
                    finsi
                    /* Cas où la configuration n'est pas encore présente. On l'insère en affectant le numéro de
                    configuration à la valeur du nœud. */
                    sinon
                        /* Cas où l'on traite une nouvelle génération. */
                        si nv_gen = 0 alors  g_tmp->num_config := nb_config_total - 1;
                        /* Cas où l'on n'est pas en train de créer une nouvelle génération. */
                        sinon  g_tmp->num_config := nb_config_total;
                        retourner nb_config_total;
                    finsi
                finsi
                /* Sinon, il s'agit simplement d'un changement de ligne. */
                sinon
                    /* Si ce changement de ligne n'existe pas encore dans l'arbre. */
                    si g_tmp->tab[0] = NULL alors
                        /* On initialise une nouvelle arête. */
                        g_tmp->tab[0] := creer_arete();
                    finsi
                    /* On déplace le pointeur permettant de parcourir l'arbre. */
                    g_tmp := g_tmp->tab[0];
                    /* On passe à la ligne suivante de la matrice. */
                    break ;
                finsi
            finsi
            /* Sinon, on verifie l'existence du noeud correspondant dans l'arbre. */
            sinon
                /* Si le noeud correspondant de l'arbre n'existe pas, on avance dans l'arbre. */
                si g_tmp->tab[c->mat[i][j]] = NULL alors
                    /* On l'initialise. */
                    g_tmp->tab[c->mat[i][j]] := creer_arete();
                finsi
                /* On déplace le pointeur permettant de parcourir l'arbre de générations. */
                g_tmp := g_tmp->tab[c->mat[i][j]]/ ;
            finsi
        finpour
    finpour
    retourner -1 ;
fin
```

Algorithme 7 : ALGO INSERTION_CONFIGURATION_SI_NÉCESSAIRE

```
/* Procédure insérant si c'est nécessaire la configuration passée en paramètre dans le tableau des configurations ainsi
que dans l'arbre de génération correspondant.
c : la configuration.
num_mere : numéro de la configuration mère. */
Procédure inserer_config_si_necessaire(configuration *c, Entier num_mere);
Données : Entier num_config_crt;
          Entier ind_1, ind_2;
début
    num_config_crt := -1;
    /* S'il s'agit de la première configuration de la génération à créer. On possède deux variables globales, l'une comptant le nombre de générations allouee, láutre le nombre de générations créées.
    NB : Quand le nombre de générations créées est impair, la dernière génération à avoir été créée est une génération paire (la première génération a pour numéro 0) et inversement pour un nombre de générations créées pair. */
    si nb_generation_allouee = nb_generation_creee alors
        cf. l'algorithme ALGO INSERTION_CONFIG_SI_NÉCESSAIRE PARTIE A
    finsi
    /* Sinon, la génération existe déjà et contient au moins une configuration. */
    sinon
        /* Recherche de la configuration créée et insertion si elle n'est pas déjà présente dans la génération en cours de création. */
        si (nb_generation_creee mod 2) = 0 alors  num_config_crt := inserer_config(gen_paire, c, 1);
        sinon  num_config_crt := inserer_config(gen_impaire, c, 1);
        si num_config_crt = -1 alors  Erreur ;
        /* Si on est en train de créer une génération paire. */
        si (nb_generation_creee mod 2) = 1 alors
            /* Calcul de l'indice de la configuration à insérer dans le tableau correspondant. */
            ind_1 := num_mere - tab_config_gen_paire[0]->num;
        finsi
        /* Sinon, on est en train de créer une génération impaire. */
        sinon
            ind_1 := num_mere - tab_config_gen_impaire[0]->num;
            reallouer_filles_config_mere_si_necessaire(num_mere);
        finsi
        /* Si la configuration vient juste d'être insérée dans la génération, cela signifie aussi qu'elle n'existe pas non plus dans le tableau des configurations. */
        si num_config_crt = nb_config_total alors
            cf. l'algorithme ALGO INSERTION_CONFIG_SI_NÉCESSAIRE PARTIE B
        finsi
        /* Sinon, la configuration est déjà insérée dans la génération et dans le tableau correspondant. Il suffit donc de mettre à jour les relations de parenté. */
        sinon
            cf. l'algorithme ALGO INSERTION_CONFIG_SI_NÉCESSAIRE PARTIE C
        finsi
    finsi
fin
```

Algorithme 8 : ALGO INSERTION_CONFIG_SI_NÉCESSAIRE PARTIE A

```
début
    /* On crée la relation de paternité au niveau de la fille. */
    c->meres[0] := num_mere;
    c->nb_meres ++;
    /* Mise à jour du nombre de générations allouées. */
    nb_generation_allouee ++;
    /* On insère la configuration dans le tableau des configurations associé en fonction de la parité de la génération. */
    si (nb_generation_creee mod 2) = 1 alors
        /* Récupération de l'indice de la configuration mère dans le tableau correspondant. */
        ind_1 := num_mere - tab_config_gen_paire[0]->num;
        /* On met à jour la relation de parenté entre les configurations. */
        tab_config_gen_paire[ind_1]->filles[0] := nb_config_total;
        tab_config_gen_paire[ind_1]->nb_filles ++;
        /* Mise en place de la configuration dans le tableau correspondant. */
        tab_config_gen_impaire[nb_config_gen_impaire] := c;
        /* Mise à jour du compteur d'éléments du tableau des configurations de génération impaire. */
        nb_config_gen_impaire ++;
    finsi
    sinon
        ind_1 := num_mere - tab_config_gen_impaire[0]->num;
        tab_config_gen_impaire[ind_1]->filles[0] := nb_config_total;
        tab_config_gen_impaire[ind_1]->nb_filles ++;
        tab_config_gen_paire[nb_config_gen_paire] := c;
        nb_config_gen_paire ++;
    finsi
    /* Mise à jour du nombre total de configurations. */
    nb_config_total ++;
    /* On initialise la nouvelle génération. Cas où l'on est en train de créer une génération paire. */
    si (nb_generation_creee mod 2) = 0 alors
        /* Initialisation d'un nouvel arbre pour la nouvelle génération. */
        gen_paire := initialiser_generation();
        /* On insère la configuration dans la génération. */
        si inserer_config(gen_paire, c, 0) = -1 alors  Erreur;
    finsi
    /* Cas où l'on est en train de créer une génération impaire. */
    sinon
        gen_impaire := initialiser_generation();
        si inserer_config(gen_impaire, c, 0) = -1 alors  Erreur;
    finsi
fin
```

Algorithme 9 : ALGO INSERTION_CONFIG_SI_NÉCESSAIRE PARTIE B

```
début
    /* On lui crée un lien de parenté avec sa configuration mère. */
    c->meres[0] := num_mere;
    c->nb_meres ++;
    /* Si l'on est en train de créer une génération impaire. */
    si (nb_generation_creee mod 2) = 1 alors
        /* Récupération de l'indice de la configuration mère dans le tableau correspondant. */
        ind_1 := num_mere - tab_config_gen_paire[0]->num;
        /* On met à jour le lien de parenté entre les configurations. */
        tab_config_gen_paire[ind_1]->filles[tab_config_gen_paire[ind_1]->nb_filles] := nb_config_total;
        tab_config_gen_paire[ind_1]->nb_filles ++;
        /* On insère la configuration dans le tableau approprié. */
        tab_config_gen_impaire[nb_config_gen_impaire] := c;
        nb_config_gen_impaire += 1 ;
    finsi
    /* Sinon, on est en train de créer une génération paire. */
    sinon
        ind_1 := num_mere - tab_config_gen_impaire[0]->num;
        tab_config_gen_impaire[ind_1]->filles[tab_config_gen_impaire[ind_1]->nb_filles] := nb_config_total;
        tab_config_gen_impaire[ind_1]-¿nb_filles ++;
        tab_config_gen_paire[nb_config_gen_paire] := c;
        nb_config_gen_paire ++;
    finsi
    /* Mise à jour du nombre total de configurations. */
    nb_config_total ++ ;
fin
```

Algorithme 10 : ALGO INSERTION_CONFIG_SI_NÉCESSAIRE PARTIE C

```
début
    /* On désalloue tout d'abord la génération nouvellement créée. */
    desallouer_configuration(c);
    /* Si on est en train de créer une génération impaire. */
    si (nb_generation_creee mod 2) = 1 alors
        ind_1 := num_mere - tab_config_gen_paire[0]->num;
        ind_2 := num_config_crt - tab_config_gen_impaire[0]->num;
        tab_config_gen_impaire[ind_2]->meres[tab_config_gen_impaire[ind_2]->nb_meres] := num_mere;
        tab_config_gen_paire[ind_1]->filles[tab_config_gen_paire[ind_1]->nb_filles] := num_config_crt;
        tab_config_gen_impaire[ind_2]->nb_meres ++;
        tab_config_gen_paire[ind_1]->nb_filles ++;
    finsi
    /* Sinon, on est en train de créer une génération paire. */
    sinon
        ind_1 := num_mere - tab_config_gen_impaire[0]->num;
        ind_2 := num_config_crt - tab_config_gen_paire[0]->num;
        tab_config_gen_paire[ind_2]->meres[tab_config_gen_paire[ind_2]->nb_meres] := num_mere;
        tab_config_gen_impaire[ind_1]->filles[tab_config_gen_impaire[ind_1]->nb_filles] := num_config_crt;
        tab_config_gen_paire[ind_2]->nb_meres ++;
        tab_config_gen_impaire[ind_1]->nb_filles ++;
    finsi
fin
```

Algorithme 11 : ALGO AB_F_MATRICE

```
/* Procédure construisant l'ab-Matrice et la f-Matrice d'une configuration dont la matrice est passée en paramètre. */
Procédure construire_abf_matrices(Entier nb_grains, Entier **mat);
Données : /* Compteur pour les lignes de niveau (nombre de lignes des matrices à créer. */)
          Entier k;
          /* Compteur pour les mots de Dyck (nombre de colonnes des matrices à créer. */)
          Entier l;
          /* Compteurs pour la matrice passée en paramètre. */
          Entier x, y;
          /* Copie de la matrice de la configuration dans un format différent, à savoir de taille égale à nb_grains. */
          Entier **nv_mat;
          /* ab-Matrice et f-Matrice. */
          Entier **ab_mat, f_mat;
début
    /* On crée la nouvelle matrice représentative de la configuration dans son nouveau format. Le but est d'obtenir une
    matrice carrée dont la taille est le nombre de grains du système (nb_grains). */
    nv_mat := creer_matrice_nouveau_format(mat);
    /* Parcours des lignes de niveaux de la plus haute à la plus basse. */
    pour k de nb_grains à 1 faire
        x := nb_grains;
        y := 1;
        /* Parcours des "mots de Dyck". */
        pour l de 1 à (2*nb_grains + 2) faire
            /* Si on est sur le premier élément du "mot de Dyck". */
            si l = 1 alors
                /* On met 'a' en 1ère position de la ligne courante de l'ab-Matrice. */
                ab_mat[nb_grains-k][l] := 97;
                /* On met 1 en 1ère position de la ligne courante de la f-Matrice. */
                f_mat[nb_grains-k][l] := 1;
            finsi
            /* Dans tous les autres cas. */
            sinon
                /* Si on est sur le dernier element du "mot de Dyck". */
                si l = (2*nb_grains + 2) alors
                    /* On met 'b' en dernière position de la ligne courante de l'ab-Matrice. */
                    ab_mat[nb_grains-k][l] := 98;
                    /* On met 0 en dernière position de la ligne courante de la f-Matrice. */
                    f_mat[nb_grains-k][l] := 0;
                finsi
                /* Autrement. */
                sinon
                    /* Si on se trouve tout en haut de la matrice. */
                    si x = -1 alors
                        ab_mat[nb_grains-k][l] := 98;
                        f_mat[nb_grains-k][l] := f_mat[nb_grains-k][l-1] - 1;
                    finsi
                    sinon
                        /* Si on tombe sur une ligne de niveau inférieure. */
                        si nv_mat[x][y] < k alors
                            ab_mat[nb_grains-k][l] := 97;
                            f_mat[nb_grains-k][l] := f_mat[nb_grains-k][l-1] + 1;
                            x − ;
                        finsi
                        sinon
                            ab_mat[nb_grains-k][l] := 98;
                            f_mat[nb_grains-k][l] := f_mat[nb_grains-k][l-1] - 1;
                            y ++ ;
                        finsi
                    finsi
                finsi
            finsi
        finpour
    finpour
fin
```

Conclusion

Ces quatre mois passés au LIAFA ont été l'occasion de travailler pour la première fois dans l'univers de la recherche en informatique fondamentale. Ils m'ont permis de collaborer avec des personnes possédant des connaissances importantes qui m'ont appris un très grand nombre de choses, non seulement aux niveaux théorique et technique du domaine lui-même mais encore en me donnant leur point de vue général de la profession qu'ils pratiquent. Comme je vais poursuivre mes études dans cette voie, j'intègre le Master de Recherche en Informatique Fondamentale de Lyon á la rentrée scolaire 2004, il était important que je puisse avoir une idée fiable de ce que représente le métier de chercheur et commencer à acquérir certains automatismes quant à la manière de présenter et de formaliser les recherches effectuées.

Ainsi, au niveau pratique, ce stage m'a permis d'améliorer et de renforcer mes capacités à résoudre des problèmes algorithmiques complexes afin d'obtenir une complexité en temps la plus faible possible ainsi que mes connaissances du langage de programmation C. En effet, en terme de programmation, les logiciels créés m'ont amené à ajuster de manière permanente un équilibre judicieux entre les complexités en temps et en mémoire tout en me faisant utiliser certaines bases de la programmation système afin de gérer les fichiers de sauvegarde. Grâce à tout cela, des petits détails du langage C qui m'étaient toujours apparus comme flous ont pu être étudiés et compris.

Par ailleurs, la partie théorique de ce stage a eu l'avantage de me faire plonger dans un "univers" baigné par les mathématiques, ce qui, je l'avoue, me faisait un peu peur au début. Toutefois, cela s'est bien passé même si, à diverses reprises, je me suis retrouvé un peu perdu. À ces moments précis, j'ai pris le temps qu'il fallait pour comprendre ce qui me paraissait flou en y rfléchissant à tête reposée, en posant des questions aux personnes qui m'entouraient. Je me suis aussi rendu compte en écrivant ce rapport que les explications et la formalisation des recherches effectuées n'était vraiment pas faciles à réaliser. Cela en fait une étape du projet qui s'est avérée très enrichissante qui m'a d'ailleurs permis d'apprendre à utiliser LaTeX, qui est un outil fort utilisé dans le monde de la recherche scientifique.

Ce stage réalisé au LIAFA sous la direction de Roberto Mantaci a finalement été pour moi le stage le plus enrichissant de tous ceux réalisés jusqu'à présent même si l'objectif ultime n'a pas été atteint, ce qui n'a pas surpris les chercheurs connaissant la difficulté de la tâche confiée. Le fait d'avoir réussi à obtenir des résultats partiels a été pour moi fort appréciable. Mon envie de devenir enseignant-chercheur a donc été confortée et même amplifiée.

Enfin, pour la satisfaction que ce stage m'a apporté et la bonne ambiance dans laquelle il s'est déroulé, il est certain que si l'occasion se présentait à nouveau, je n'hésiterais pas une seconde et retravaillerais avec joie au sein de l'équipe d'algorithmique du LIAFA de Paris.

Références

[Bak97] P. Bak. *How nature works - The science of SOC*. Oxford university press, 1997.

[Bri73] T. Brilawski. The lattice of integer partitions. *Discrete Mathematics*, 6 :210–219, 1973.

[BS95] A.-L. Barabási and H.E. Stanley. *Fractal Concepts in Surface Grow*. Cambridge university press, 1995.

[BT98] P. Bak and C. Tang. *Journal of Geophysics*, B94 :15635, 1998.

[BTW87] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality : An explanation of 1/f noise. *Physics Review Letters*, 59 :381, 1987.

[CNLD96] B.A. Carreras, D.E. Newman, V.E. Lynch, and P.H. Diamond. A model realization for soc for plasma confinment. *Physics of Plasmas*, 3(8), 1996.

[Dur97] J. Duran. *Sables, poudres et grains*. Eyrolles sciences, 1997. Préface de P.G. de Gennes.

[Eri93] K. Eriksson. *Strongly Convergent Games and Coexter Groups*. PhD thesis, Kungl Tekniska Hogskolan, Sweden, 1993.

[GK93] E. Goles and M.A. Kiwi. Games on line graphs and sand piles. *Theoretical Computer Science*, 115 :321–349, 1993.

[GMP02] E. Goles, M. Morvan, and H.D. Phan. Sand piles and order structure of integer partitions. *Discrete Applied Mathematics*, 117 :51–64, 2002.

[Jen98] H.J. Jensen. *Self-Organised Criticality*. Cambridge university press, 1998.

[LH91] E. Lu and R. Hamilton. *Astrophysics Journal*, 380 :L89, 1991.

[LMMP01] M. Latapy, R. Mantaci, M. Morvan, and H.D. Phan. Structure of some sand piles models. *Theoretical Computer Science*, 262 :525–556, 2001.

[Tan93] C. Tang. Self-organised criticality. *OCPA Newsletter*, 1993.

[Tur97] D.L. Turcotte. *Fractals and Chaos in Geology and Geophysics*. Cambridge university press, 1997.

[WS93] Z. Wang and D. Shi. *Physics Review B*, 48 :9782, 1993.