THÈSE

présentée à

L'UNIVERSITÉ DE LA MÉDITERRANÉE
AIX-MARSEILLE II

École doctorale de Mathématiques et Informatique

par

Ian Gambini

pour obtenir le grade de

DOCTEUR ÈS SCIENCES

spécialité :

INFORMATIQUE

Quant aux carrés carrelés

---

soutenue le : 18 Décembre 1999

devant le jury composé de :

M. Alain Colmerauer	Directeur de thèse
M. Philippe Flajolet	Examinateur
M. Pascal van Hentenryck	Rapporteur
M. Claude Kirchner	Rapporteur
M. Jean-François Maurras	Examinateur

Rapporteurs et membres du jury

Alain Colmerauer, Directeur de thèse
Professeur à l'Université de la Méditerranée
Département d'Informatique
Faculté des Sciences de Luminy
163, Avenue de Luminy
13288 Marseille Cedex 9
France

Philippe Flajolet, Examinateur
Directeur de Recherche
INRIA Rocquencourt
Domaine de Voluceau
BP 105
78153 Le Chesnay Cedex
France

Pascal van Hentenryck, Rapporteur
Professeur
(aussi Adjunct Associate Professor at Brown University (USA))
Département d'Ingénierie Informatique,
Université catholique de Louvain,
2, Place Sainte-Barbe,
B-1348, Louvain-la-Neuve
Belgique

Claude Kirchner, Rapporteur
Directeur de Recherche
LORIA & INRIA
615 rue du Jardin Botanique
BP 101
54602 Villers-lès-Nancy
France

Jean-François Maurras, Examinateur
Professeur à l'Université de la Méditerranée
Département d'Informatique
Faculté des Sciences de Luminy
163, Avenue de Luminy
13288 Marseille Cedex 9
France

Remerciements

Il me sera très difficile de remercier tout le monde car c'est grâce à l'aide de nombreuses personnes que j'ai pu mener cette thèse à son terme.

Je voudrais tout d'abord remercier grandement mon directeur de thèse, Alain Colmerauer, pour toute son aide. Je suis ravi d'avoir travaillé en sa compagnie car outre son appui scientifique, il a toujours été là pour me soutenir et me conseiller au cours de l'élaboration de cette thèse. Je remercie également Colette qui a su inspirer le titre de cette thèse...

Claude Kirchner et Pascal Van Hentenryck m'ont fait l'honneur d'être rapporteurs de ma thèse, ils ont pris le temps de m'écouter et de discuter avec moi. Leurs remarques m'ont permis d'envisager mon travail sous un autre angle. Pour tout cela je les remercie.

Je tiens à remercier Jean-François Maurras pour avoir accepté de participer à mon jury de thèse et pour sa participation scientifique ainsi que le temps qu'il a consacré à ma recherche.

Je remercie également Philippe Flajolet pour l'honneur qu'il me fait d'être dans mon jury de thèse.

Au cours de ces années j'ai fait parti de l'équipe « Contraintes, Algorithmes et Combinatoire » au sein du LIM. Les discussions que j'ai pu avoir durant les réunions d'équipe ou en dehors avec Noëlle Bleuzen, Alain Guenoche, Jean-François Pique, Victor Chepoï, Jean-Luc Massat et Yann Vaxes m'ont beaucoup apporté. Je remercie donc toutes ces personnes. Je tiens à remercier particulièrement Michel Van Caneghem pour toutes nos discussions et ses conseils qui m'ont accompagné tout au long de mon cursus à Luminy.

Il m'est impossible d'oublier Guylaine Vincent pour son aide précieuse pour ma recherche bibliographique. Elle a toujours fait tout son possible pour m'aider. Je tiens aussi remercier Sylvie Calabrese, Solange Panattoni et Annie Cobalto pour leur gentillesse et leur aide.

Durant ma thèse j'ai aussi effectué de nombreux enseignements à l'IUT GTR et je remercie Kader Betari, Patrick Girard et Eric Soccorsi pour leur aide et pour avoir fait en sorte de me laisser du temps libre afin de terminer la rédaction de ma thèse. Un grand merci aussi pour leur aide et leurs encouragements à Mylène Malpas, Hervé Dallaporta, Tin Nguyen, Arnaud Février, Rolland Depeyre, Valérie Mejean, André Costa et Frédéric Guidicelli.

Ayant mon bureau au sein de l'ESIL, je remercie Traian, Léon, Touraïvane, Gérard qui m'ont entouré et m'ont conseillé, ainsi que tous les thésards pour m'avoir supporté tous les jours depuis plusieurs années, même si ce n'est pas encore fini ! : Bruno, David, Bich, Nicolas , Pédro, Cédric, Olivier, Stéphane, Mathieu, Christian, Jianyang, Malika, Nabil. Ils m'ont beaucoup aidé et sont devenus des amis à qui je souhaite tout le courage qu'ils m'ont apporté.

Je remercie toutes les personnes avec qui j'ai partagé mes études et notamment ces années de thèse : Stéphane et Patricia qui m'ont permis de m'échapper de temps en temps dans la France profonde ; Noël et Vincent pour les Irish coffee que j'ai pu gagner en jouant au Badminton avec eux ; Myriam et Sébastien pour leur gentillesse et les délicieux bonbons au goût d'érable made in Canada ; Laurence, Christophe, Boïde, Nicolas, Eric, Fabienne, Viet, Cyril, Pascal avec qui j'ai du passer des heures à la cafétéria pour refaire le monde ; Denise, les 2 Françoise, Solange, Edith, Stéphan, Stéphane dit «le Djaoui», Claude, Christine, Jeanine, Pas-

cal et Patrick pour leur inévitable rituel du midi et leur animation dans la cafétéria, qui m'ont soutenu jusqu'au bout.

Heureusement que mes parents, mes frères et mes amis sont la pour me changer les idées. Ils ont tous cru en moi et ouf ! maintenant j'y suis ! Alors merci à vous tous, Itziar, Raymond, Eric, Christine et Louis, Claudie et François, Patrick, Christian, Laurent et Yolaine, Christine et Yann, Laurent et Hélène, Jérome et Carole et Bastien, Marion, Olivia et Thomas, Cathy et Antoine, Nathalie et Hervé, Henri, Denis, Olivier et Catherine, Hélène, Florette, Amanda, Monia,...

Mes derniers remerciements vont à Amandine qui a tout fait pour m'aider, qui m'a soutenu et surtout supporté dans tout ce que j'ai entrepris.

J'en oublie certainement encore et je m'en excuse.

Encore un grand merci à tous pour m'avoir conduit à ce jour mémorable.

Je dédie cette thèse à Karine.

Résumé

Cette thèse est consacrée au calcul de découpages d'un carré en carrés de tailles différentes.

Tout d'abord nous reprenons la méthode classique : énumération de graphes planaires représentant les emplacements des carrés les uns par rapport aux autres et pour chaque graphe résolution d'un système linéaire donnant les dimensions des carrés sous forme de nombres rationnels. La difficulté essentielle est de travailler en précision infinie. Pour celà nous étudions différentes méthodes qui permettent de contrôler la taille d'un dénominateur commun à tous les nombres rationnels manipulés. Pour un entier n donné nous obtenons ainsi toutes les décompositions en n carrés. Notre programme retrouve les solutions connues pour n entre 21 et 26.

En fait ce problème de découpage est un problème en nombres entiers : les tailles des différents carrés sont toutes commensurables (dans des rapports rationnels). Il est donc possible de calculer les découpages par énumération d'entiers représentant des tailles de carrés. C'est l'objet du reste de la thèse.

Nous établissons tout d'abord quelques propriétés concernant les dimensions entières des carrés : entre autre, les carrés dans les coins sont de taille supérieure à 8 et ceux sur les bords de taille supérieure à 4. Ceci nous permet de développer un algorithme opérationnel qui, étant donné la taille d'un carré, trouve tous ses découpages possibles. Nous arrivons ainsi à découper des carrés de taille allant jusqu'à 130 et retrouvons les découpages minimaux bien connus et notamment celui en 21 carrés.

Nous développons alors un deuxième algorithme beaucoup plus efficace mais incomplet : il énumère des découpages particuliers. Nous obtenons plus de 30000 solutions parmi lesquelles figure toujours le fameux découpage en 21 carrés. Avec quelques modifications cet algorithme nous fournit des solutions à deux problèmes connexes : la décomposition en carrés de tailles distinctes d'un cylindre et aussi d'un tore. Dans le premier cas le cylindre est de hauteur et de circonférence égale et les carrés sont formés de deux arcs de cercle et de deux segments de même longueur. Dans le deuxième cas, les côtés des carrés sont des arcs de cercle de dimensions angulaires égales.

Mots-clefs

Décomposition de carrés. Découpages de carrés. Carrés carrelés. Carrelages.

Abstract

About squared squares

This thesis is devoted to the calculation of decompositions of a square into distinct sized squares.

First of all, we take up the classical method : enumeration of planar graphs representing the position of squares in relation to the others and for each graph, resolution of linear equations giving the size of each squares in a rational form. The essential difficulty is to work using infinite precision. For this, we study different methods allowing us to control the size of a common denominator of all rational numbers manipulated. For a given integer n we obtain every decompositions using n squares. Our program finds all the known solutions for n between 21 and 26.

In fact, this problem of decomposition is an integer problem : the sizes of each square are commensurable (they have rational ratio). So it is possible to find decompositions by enumerating integers representing the sizes of the squares. This is the purpose of the rest of this thesis.

First of all, we establish some properties concerning the integer dimensions of the squares : among other things, squares at the corner have a size greater than 8 and those at the border have a size greater than 4. Those properties allow us to develop an operational algorithm which, given the size of a square, find all its decompositions. Like this we can decompose squares of size up to 130 and we find the well known minimal decompositions, among others we find the one using 21 squares.

Then we develop a second algorithm a lot more efficient but incomplete : decompositions are of a particular type. We obtain more than 30,000 solutions in which we still find the decomposition using 21 squares. With some modifications, this algorithm allows us to find solutions of two closely related problems : the decomposition using distinct sizes squares of a cylinder and also of a tore. In the first case, the cylinder have the same height than his circumference and the squares are made of two arc of a circle and of two segments having the same length. In the second case, squares are made of 4 arcs of a circle having equal angular dimensions.

Keywords

Squaring the square. Squared Squares. Decomposition of squares.

Table des matières

Chapitre 1

Introduction

Le problème qui nous intéresse est le découpage d'un carré en plusieurs carrés de tailles distinctes. C'est un cas particulier du découpage d'un rectangle en carrés distincts, problème plus ancien, déjà abordé en 1903 par Max Dehn [20].

La figure 1.1 représente le plus célèbre découpage d'un carré en carrés de tailles toutes distinctes. Ici le nombre de carrés découpés est le plus petit possible. Ce découpage a été trouvée en 1978 par A.J.W. Duijvestijn [22], ce qui est relativement récent, et ceci grâce à l'utilisation d'ordinateurs.

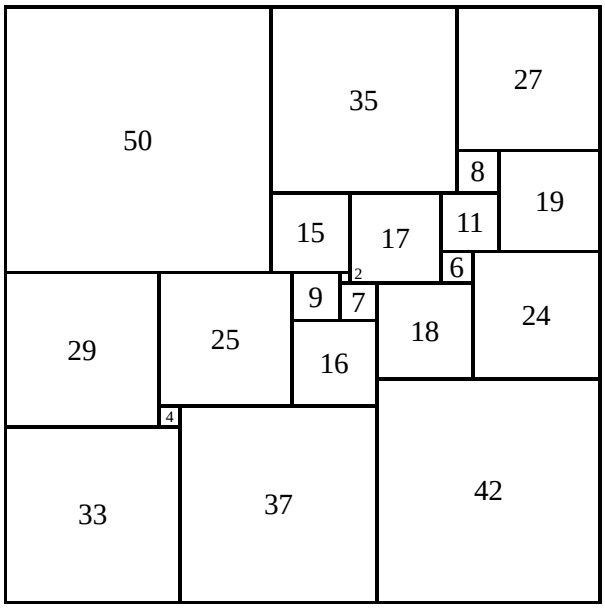


FIG. 1.1 – Le carré parfait d'ordre 21 $(21 : 112 \times 112)$.

Le code de Bouwkamp [6] de cette décomposition en 21 carrés est :

(50,35,27) (8,19) (15,17,11) (6,24) (29,25,9,2) (7,18) (16) (42) (4,37) (33)

D'une façon générale le code de Bouwkamp d'une décomposition de carré ou de rectangle consiste en la liste des carrés énumérés de haut en bas et de gauche à droite. Pour que ce code soit unique (à cause des 8 symétries possibles), on impose les conditions suivantes :

– les tailles doivent être entières et les plus petites possibles ;
– le côté le plus large de la décomposition doit être horizontal ;
– le carré du coin en haut à gauche doit être plus grand que ceux des trois autres coins ;
– dans le cas d'une décomposition de carré, on impose aussi que l'élément situé le plus en

haut et juste à la droite du carré au coin en haut à gauche doit être plus grand que le carré situé le plus à gauche et juste en dessous du carré au coin en haut à gauche.

Lors de l'écriture, on met des parenthèse pour regrouper les carrés se trouvant côte à côte et ayant le côté haut aligné. Ce code permet une écriture unique et compacte de chaque solution.

Historique

Avant de trouver des décompositions de carrés, plusieurs personnes ont essayé de trouver des décompositions de rectangles en carrés distincts. Le tout premier rectangle *parfait* fut découvert par Z. Moron en 1925. Ce rectangle est aussi le rectangle parfait d'ordre minimal.

Un découpage est dit *parfait* lorsque tous les carrés de sa décomposition ont une taille différente et son *ordre* correspond au nombre d'éléments (de carrés) utilisés. Nous noterons $n : x \times y$ une décomposition d'ordre n et de taille $x \times y$.

Le problème du découpage d'un carré en carrés distincts est longtemps resté sans réponse bien que M. Kraitchik [39], mathématicien Russe, publia en 1930 une communication de Lusin démontrant qu'il était impossible de découper un carré avec un nombre fini de carrés distincts.

Malgré cela, le premier découpage parfait de carré fut découvert par R. Sprague en 1939 [44]. Depuis, de nombreuses autres solutions ont été trouvées et on a longtemps cherché une solution d'ordre minimale. En 1962, C.J. Bouwkamp, A.J.W. Duijvestijn et P. Medema [21] ont montré qu'il n'existait pas de décomposition parfaite d'un carré en carrés d'ordre inférieur à 21. En 1948, un carré parfait d'ordre 24 fut trouvé par T.H. Willcocks [49] (figure 1.2), suivi en 1964 par J.S. Wilson [46] avec un carré parfait d'ordre 25. En 1967, cinq solutions de carrés parfaits d'ordre 25 furent publiées par J.S. Wilson dans sa thèse [50]. Ensuite, trois autres carrés parfaits d'ordre 25 furent obtenus par P.J. Federico en 1978 [28]. Finalement, A.J.W. Duijvestijn [22] obtint une solution d'ordre 21 (figure 1.1) trouvée le 22 Mars 1978 avec l'aide d'un ordinateur DEC-10. Cette solution est l'unique solution d'ordre minimale, aux symétries près.

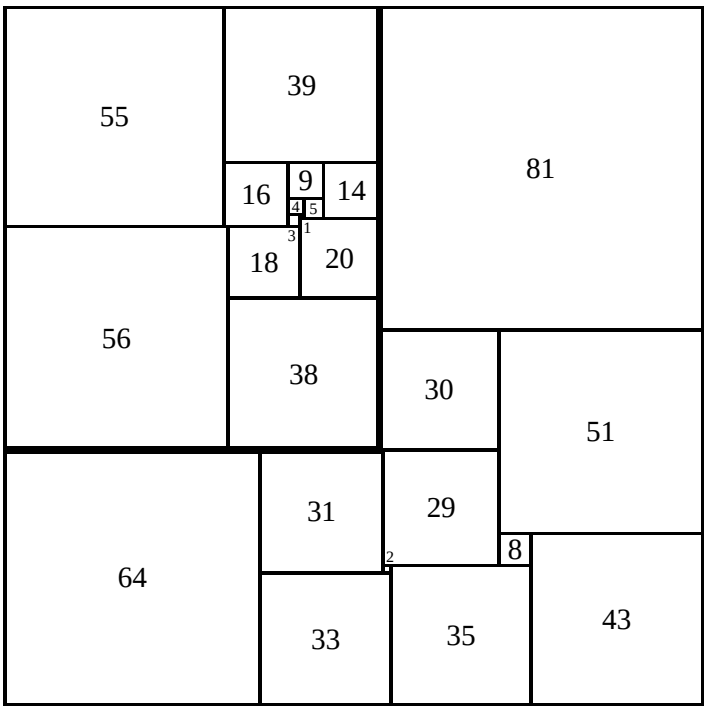


FIG. 1.2 – Carré parfait composé d'ordre 24 $(24 : 175 \times 175)$.

Le même type de résultats ont été trouvés par A.J.W. Duijvestijn, P.J. Federico et P. Leeuw [25] pour des découpages *composés*, dans lesquels un sous ensemble strict de carrés forme un rectangle. La figure 1.2 nous donne la seule décomposition parfaite *composée* d'ordre minimale (24 carrés), hors symétries.

A l'heure actuelle, les résultats que nous avons pu trouver dans la littérature sont répertoriés dans le tableau 1.1 pour les décompositions parfaites de rectangles en carrés et le tableau 1.2 pour les décompositions parfaites de carrés en carrés.

Ordre	*Nombre*	*Ordre*	*Nombre*
9	2	14	744
10	6	15	2 609
11	22	16	9 016
12	67	17	31 426
13	213	18	110 381

TAB. 1.1 – Nombre de rectangles parfaits simples

Ordre	*Nombre*
21	1
22	8
23	12
24	26
25	160
26	441

TAB. 1.2 – Nombre de carrés parfaits simples

Nous ne savons pas quel est le type des machines qui furent utilisées pour résoudre ces calculs, ni même les temps de calcul nécessaires pour les obtenir. Ceux-ci sont relativement récents mais il faut tenir compte de la montée en puissance des ordinateurs ces dernières années qui est multipliée par deux environ tous les 18 mois.

Pour se rendre compte de l'évolution du problème, nous avons répertorié l'ensemble des principales découvertes dans le tableau 1.3.

La plupart des découvertes avant 1960 ont été obtenues manuellement, mais après cette date, les ordinateurs ont permis d'effectuer de nombreuses recherches de solutions par énumération. La technique utilisée sur ces ordinateurs est basée sur l'approche manuelle que nous décrivons dans la suite.

Une définition plus formelle du problème

Quoique nous en ferons pas usage, il est possible d'exprimer notre problème sous une forme mathématique. Nous souhaitons décomposer un rectangle de côté $a \times b$ en n carrés de tailles différentes $\{a_1, ..., a_n\}$. La position d'un carré i est décrite par ses coordonnées (x_i, y_i) du coin inférieur gauche en prenant le coin inférieur gauche du grand carré comme origine. Donc tout carré débute en (x_i, y_i) et termine en $(x_i + a_i, y_i + a_i)$.

On veut donc trouver $x_1, y_1, a_1, ..., x_n, y_n, a_n$ tels que :

Date	***Nom***	***Evènement***
1903	Max Dehn [20]	Preuve qu'un découpage de rectangle de taille entier en un nombre fini de rectangles ne peut se faire qu'avec des rectangles rationnels.
1925	Zbigniew Moron [40]	Premier rectangle parfait $9 : 33 \times 32$ (celui d'ordre minimal).
1930	Lusin [39]	Preuve qu'il est impossible de découper un carré avec un nombre fini de carrés distincts.
1939	R. Sprague [44]	Premier découpage parfait de carré (ordre 55).
1940	Reichert & Toepkin [41]	Preuve qu'un rectangle ne peut se décomposer en moins de 9 carrés.
1940	R.L. Brooks et al. [13]	Carré parfait composé d'ordre 26 ($26 : 608 \times 608$)
1948	T.H. Willcocks [49]	Carré parfait d'ordre 24.
1960	C.J. Bouwkamp, A.J.W. Duijvestijn, P. Medema [7]	Enumération de tous les graphes de 19 arêtes et tous les rectangles jusqu'à 15 éléments.
1961	Martin Gardner [31]	Carrés composés d'ordres 24 et 39.
1962	A.J.W. Duijvestijn et al. [21]	Il n'existe pas de décomposition de carré en carrés parfaits d'ordre inférieur à 21.
1964	C.J. Bouwkamp, A.J.W. Duijvestijn, J. Haubich [8]	Catalogue de rectangles parfaits d'ordres 9 à 18.
1964	J.S. Wilson [46]	Carré parfait d'ordre 25.
1967	J.S. Wilson [50]	5 carrés parfaits d'ordre 25.
1978	P.J. Federico [28]	3 carrés parfaits d'ordre 25.
1978	A.J.W. Duijvestijn [22]	Le carré parfait d'ordre minimal ($21 : 112 \times 112$).
Jan 1992	C.J. Bouwkamp [9]	5 carrés parfaits d'ordre 25.
Nov 1992	C.J. Bouwkamp, A.J.W. Duijvestijn [10]	Tous les carrés parfaits simples d'ordre 21 à 25.
Déc 1994	C.J. Bouwkamp, A.J.W. Duijvestijn [11]	Catalogue des carrés parfaits d'ordre 21 à 26.

TAB. 1.3 – Dates des principales découvertes

1. Toutes les tailles des carrés doivent être différents.

$$\forall i \in \{1, ..., n-1\}, \forall j \in \{i+1, ..., n\}, a_i \neq a_j$$

2. Tout point du rectangle $a \times b$ est recouvert par un seul carré.

$$\forall x \in [0, a), \forall y \in [0, b), |\{i \in 1..n \,|x_i \leq x < x_i + a_i \; et \; y_i \leq y < y_i + a_i \}| = 1$$

3. Uniquement le rectangle $a \times b$ doit être recouvert.

$$\sum_{i=1}^{n} a_i^2 = ab$$

D'autres propriétés de ce problème peuvent être intéressantes comme par exemple le fait que la somme de toutes les tailles des carrés se trouvant sur une ligne horizontale (resp. verticale) est égales à a (resp. b). On peut exprimer ces propriétés de la façon suivante :

$$\forall y \in [0, b), \sum_{i=1}^{n} (y \in [y_i, y_i + a_i)) \times a_i = a$$

$$\forall x \in [0, a), \sum_{i=1}^{n} (x \in [x_i, x_i + a_i)) \times a_i = b$$

Ces propriétés sont intéressantes lorsque l'on cherche à placer les carrés d'une décomposition dont on connaît leurs tailles [1, 17, 36, 51]. Celles-ci sont surtout très pratiques en programmation par contraintes.

Organisation de la thèse

Mis à part l'introduction et la conclusion, cette thèse comporte cinq parties numérotées de 2 à 6.

Dans la partie 2, nous étudions la méthode de résolution classique. Pour cela, nous effectuons tout d'abord une énumération de graphes planaires ayant tous les nœuds de degré 3. Ces graphes représentent l'emplacement des carrés les uns par rapport aux autres. Chaque graphe est ensuite considéré comme un réseau de résistances sur lequel il nous faut résoudre les équations de Kirchhoff pour trouver les tailles des carrés correspondants. Pour cela nous faisons une étude de différentes méthodes de résolutions en précision infinie.

La partie 3 est dédiée à l'étude de diverses propriétés. Nous montrons tout d'abord qu'il s'agit d'un problème aux nombre entiers, c'est-à-dire que toute décomposition en carrés distincts peut s'exprimer en utilisant uniquement des tailles entières. Nous montrons alors que certaines tailles de carrés sont interdites dans les coins et sur les bords.

Dans la partie 4 nous utilisons les précédentes propriétés pour aboutir à un algorithme d'énumération exhaustif opérationnel. Ainsi, pour une taille donnée, il nous est possible de trouver toutes les décompositions d'un carré de cette taille en carrés distincts. Cette méthode permet aussi de trouver des découpages de rectangles ayant pour plus grand côté la taille donnée.

La partie 5 est consacrée à l'étude d'un algorithme d'énumération sélectif. A partir de remarques sur les décompositions connues, nous établissons des règles de construction simples qui nous permettent de trouver de nombreuses solutions.

Dans la partie 6, nous nous intéressons à des problèmes connexes à celui de la décomposition d'un carré. Nous étudions tout d'abord la décomposition d'un cylindre, puis celle d'un tore en carrés (arrondis) de tailles distinctes. Pour cela nous utilisons l'algorithme d'énumération sélectif précédent. Enfin nous montrons qu'il est impossible de découper un cube en plusieurs cubes de tailles distinctes.

Plateforme de travail

Les résultats de cette thèse d'informatique sont essentiellement expérimentaux. Pour bien les juger il faut connaître les performances de l'ordinateur utilisé. Voici ce que fut notre plateforme de travail :

```
    Processeur :  Pentium Pro
     Fréquence :  200 Mhz
Mémoire Cache :  256 KB
       Système :  Linux 2.0.36
  Compilateur :  gcc 2.95.2
```

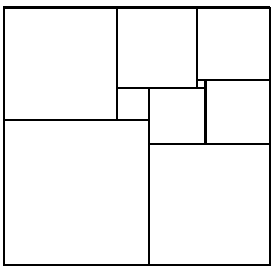
Chapitre 2

Approche classique du problème de décomposition

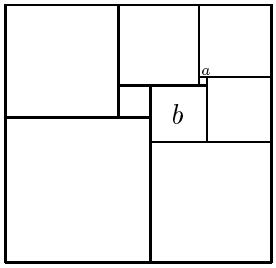
2.1 Résolution manuelle

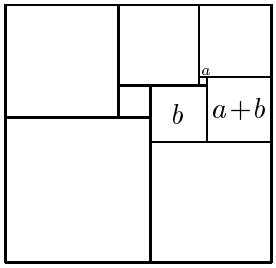
Au tout début, les choses se faisaient manuellement et de manière très intuitive. Premièrement, on essayait de trouver une solution en dessinant un rectangle sur une feuille de papier qu'on subdivisait vaguement en plusieurs carrés. Pour savoir si le dessin était bien une solution il fallait trouver les différentes tailles des carrés. Pour cela on utilisait des variables algébriques représentant les tailles de chacun des carrés et on fabriquait les équations permettant de résoudre le problème. Une fois le système d'équations résolu, les tailles des différents carrés étaient connues et on pouvait savoir si cette décomposition était bien une décomposition parfaite (tous les carrés de tailles différentes) de carré ou de rectangle.

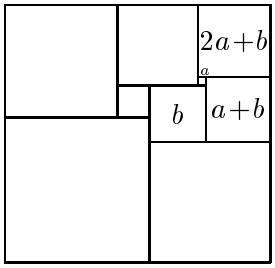
Essayons de résoudre le problème sur l'exemple suivant :



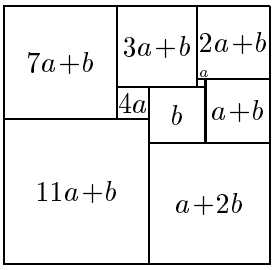
Prenons ce rectangle divisé en carrés, nous allons essayer de savoir s'il s'agit d'une solution correcte ou non. Tout d'abord, nous devons essayer d'avoir un minimum d'inconnues dans notre résolution. Pour démarrer, une méthode qui marche correctement part du plus petit carré en lui donnant la taille a et en donnant au carré le plus petit qui le touche la valeur b. Maintenant, il faut essayer de trouver les tailles des différents carrés en fonction de a et b. Avec un peu d'habitude on trouve rapidement les tailles suivantes :



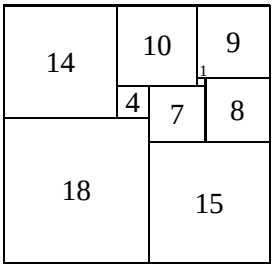




En continuant ainsi, on obtient le résultat suivant :



Cette décomposition nous donne donc un rectangle de $(18a + 2b) \times (12a + 3b)$. Il nous reste donc à définir a et b. Si on regarde attentivement le dessin, il y a une équation que nous n'avons pas utilisée, qui est $(4a) + (11a + b) = (b) + (a + 2b)$, ce qui nous permet de supprimer la variable b puisque notre équation nous donne $b = 7a$. Ainsi, notre dessin n'a plus qu'un seul degré de liberté. Une fois ceci fait, il ne nous reste plus qu'à donner une valeur à a. Prenons $a = 1$, ainsi toutes nos valeurs seront entières et les plus petites possibles. Maintenant nous savons si le dessin du départ est une solution ou non puisque nous pouvons dessiner la figure avec les tailles réelles.



Nous avons donc obtenu une décomposition parfaite de rectangle d'ordre 9 et de taille 32×33.

En utilisant cette méthode, il est possible de trouver quelques résultats, mais cette technique est fastidieuse car manuellement longue. De plus, lorsque le nombre de carrés de la décomposition augmente, on tombe souvent sur des carrés de tailles identiques. Sachant que le nombre de carrés minimum qu'il faut pour trouver la décomposition parfaite d'un carré est 21 alors qu'il faut au minimum 9 carrés pour décomposer un rectangle, on a assez peu de chances de trouver une solution de carré manuellement.

Nous allons étudier dans la suite le développement de cette méthode de manière automatique et ainsi calculable totalement avec un ordinateur. La majorité des solutions actuelles ont été calculées en utilisant cette approche de résolution. Une autre technique consiste à effectuer des transformations ou des combinaisons de solutions existantes [9, 12].

2.2 Analogie avec les graphes planaires

Si on veut généraliser pour rendre automatique la procédure manuelle précédente, il faut pouvoir définir l'emplacement des carrés les uns par rapport aux autres. Pour cela, la représentation sous forme de graphe est sans doute la plus pratique.

Avant de rentrer dans les détails de la construction, la figure 2.1 nous donne un exemple de représentation de la solution avec 21 carrés (figure 1.1, page 2).

Sur ce graphe, chaque arête correspond à un carré de la décomposition, c'est pour cette raison que nous avons placé la taille des carrés sur chacune des arêtes correspondantes. De même, on peut se poser la question de la correspondance entre les nœuds du graphe et la solution. Si on «calque» ce graphe sur la décomposition correspondante, on peut remarquer

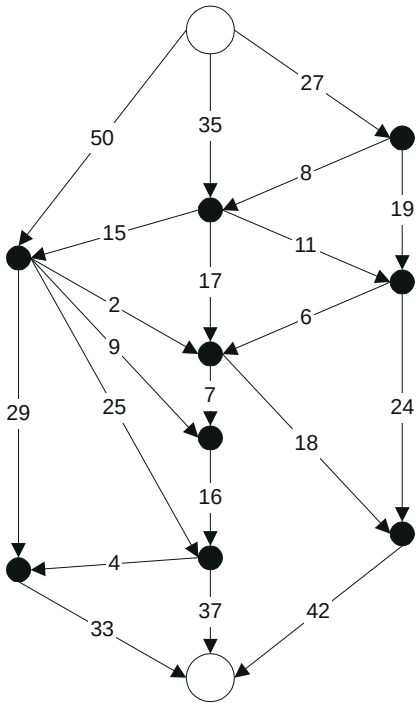


FIG. 2.1 – Graphe issu de la décomposition de la figure 1.1

que chaque nœud du graphe correspond à un segment de droite horizontal. Tout segment de droite horizontal dans la décomposition donnera un nœud dans le graphe correspondant et tout carré nous donnera une arête. Le plus simple est encore une fois de donner un exemple sur la décomposition simple qu'est le rectangle parfait $9 : 32 \times 33$ (figure 2.2).

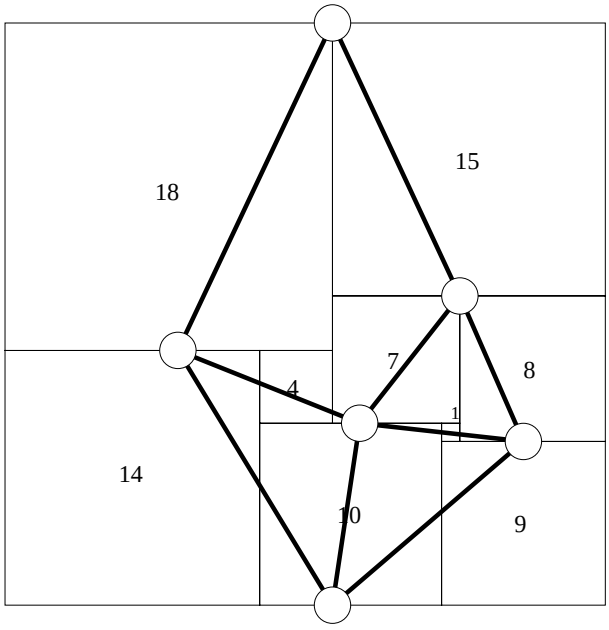


FIG. 2.2 – Décomposition et graphe associé du rectangle 32×33 d'ordre 9.

Les graphes issus de décompositions ont des caractéristiques très intéressantes :
– A chaque arête correspond un carré. Il y a donc autant d'arêtes que de carrés placés.

- La valeur portée sur chacun des arcs du graphe correspond à la taille du carré correspondant dans la décomposition.
- Les nœuds du graphe correspondent aux segments de droite horizontaux de la décomposition.
- Les graphes ainsi trouvés sont des graphes planaires.
- Si on ajoute une arête entre la source et le puits, on obtient un graphe planaire dont tous les nœuds sont de degré au moins 3. Si un nœud est de degré 2, cela implique que deux carrés ont la même taille et la décomposition ne serait donc pas *parfaite*.

Toute décomposition parfaite d'un rectangle en carrés peut donc se représenter sous la forme d'un graphe planaire ayant tous ses nœuds de degré 3, ce qui ramène le problème de décomposition à un problème d'énumération de graphe.

2.2.1 Analogie avec un réseau de résistances

Nous allons voir comment il est possible de retrouver les tailles des carrés de la décomposition à partir du graphe planaire associé. Une façon originale de trouver les tailles des carrés est de faire la remarque suivante :

Découpons dans une plaque de métal de résistivité ρ et d'épaisseur e la forme de notre décomposition. Ensuite, on isole les bords verticaux de la construction en laissant les bords horizontaux en contact. On obtient ainsi un circuit de résistances dans lequel il est possible de faire quelques remarques.

Appelons L la largeur et l la longueur de notre découpage. La surface de conduction S est $S = e \times L$ et la résistance totale est donc $R = \rho \times \frac{l}{S} = \rho \times \frac{l}{e \times L}$. Pour chacun des carrés de la décomposition, puisque la largeur (L_i) est égale à la longueur (l_i) on a $R_i = \frac{\rho}{e}$. On peut donc voir ici que quelle que soit la taille d'un carré, sa résistance sera constante. Pour simplifier, fixons une épaisseur $e = \rho$, dans ce cas, tous les carrés auront une résistance de 1Ω. Maintenant que nous avons fabriqué un réseau de résistances unitaires à partir de notre décomposition, nous allons chercher une méthode pour retrouver la taille de chacun des carrés.

Si sur notre montage électrique, nous appliquons une tension égale à l volts sur toute la longueur, puisque notre plaque a une surface de conduction constante tout le long de sa longueur, la variation de courant varie linéairement sur notre plaque, donc tout carré ayant une longueur l_i se verra forcément appliquer une différence de potentiel de l_i volts à ses bornes. On a donc égalité entre la taille d'un carré et la tension à ses bornes.

Donc, si on considère le graphe précédent comme un réseau électrique dont toutes les arêtes sont des résistances unitaires, si on applique un courant sur sa longueur et si on résout les équations de Kirchhoff, nous allons obtenir comme tension (ou comme intensité, puisque toutes les résistances sont unitaires) aux bornes de chaque résistance, la valeur correspondant aux tailles des carrés de la décomposition.

2.3 Génération des graphes planaires

Maintenant que nous savons comment trouver les tailles des carrés d'une décomposition à partir de son graphe associé, nous allons chercher à énumérer tous les graphes planaires ayant tous les nœuds de degré 3 et ceci pour un nombre donné d'arêtes.

W.T. Tutte [48] donne un théorème permettant l'énumération de graphes planaires : Considérons l'ensemble S_B de graphes planaires ayant B arêtes. Soit s un élément de S_B et s' son dual. Alors, si s n'est pas une roue, alors au moins un des graphes s ou s' peut être construit à partir d'un élément t de S_{B-1} par addition d'une arête joignant deux nœuds de t. Une roue est un graphe planaire ayant un nombre pair $2a$ de nœuds, avec un nœud de degré a et $2a - 1$

nœuds de degré 3. Le degré d'un nœud est le nombre d'arêtes joignant celui-ci. A l'aide de ce théorème, l'ensemble S_B peut être construit à partir de S_{B-1}.

Le problème de cette méthode provient du codage des graphes générés car il faut pouvoir savoir à tout moment quelles sont les arêtes que l'on peut ajouter sans perdre la planarité de notre graphe. L'utilisation d'une matrice des nœuds adjacents ne permet pas facilement de savoir si un graphe est planaire ou non. Une nouvelle représentation des graphes doit être utilisée pour s'accorder au mieux à notre méthode d'énumération.

Supposons que le graphe soit dessiné sur une sphère. Les nœuds sont étiquetés arbitrairement de 1 à N, où N est le nombre de nœuds. Le bord d'une face contient un ensemble de nœuds. Une codification de la face est obtenue en parcourant le bord de la face dans le sens positif. Il existe donc autant de codifications de faces que de nœuds sur cette face puisque nous n'avons pas fixé le nœud de départ. A partir de là, la codification du graphe est par exemple, la séquence des codes de ses faces séparées par des zéros. J.D. Skinner [43] grâce à cette représentation peut énumérer les graphes planaires en énumérant les différents codes possibles pouvant être générés par des graphes planaires.

Cette méthode nous a semblé plutôt complexe à mettre en œuvre et nous avons préféré utiliser une approche plus naturelle pour effectuer cette énumération de graphe. C'est ce que nous allons décrire dans la suite.

2.3.1 Initialisation

Il est possible d'utiliser une méthode d'énumération de graphes planaires relativement simple. Supposons que nous connaissons le nombre de carrés sur le côté gauche de notre construction. Nous allons donc démarrer notre algorithme d'énumération avec une configuration initiale identique à la figure 2.3 (pour 3 carrés au bord).

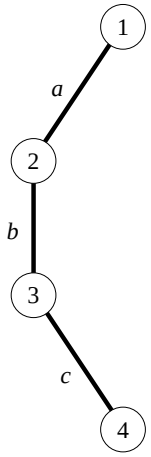


FIG. 2.3 – Initialisation de l'énumération de graphes planaires pour 3 carrés au bord

Dans cette figure, le nœud 1 est appelé la *source* et le nœud 4 est appelé le *puits*. Les 3 arêtes a, b et c correspondent aux 3 carrés constituant le bord de la décomposition.

A partir de cette forme et à chaque étape, nous allons ajouter un nouveau chemin entre deux nœuds situés sur le chemin le plus à droite entre la source et le puits, avec ou sans ajout de nouveaux nœuds. La figure 2.4 nous donne trois exemples pour la création d'un chemin entre le nœud 1 et le nœud 3.

Nous allons donc énumérer tous les chemins qu'il est possible de rajouter ainsi. Cette énumération semble très lourde mais nous avons une contrainte importante. En effet, puisque nous construisons notre graphe planaire en ajoutant toujours des chemin à droite, dans l'exemple de

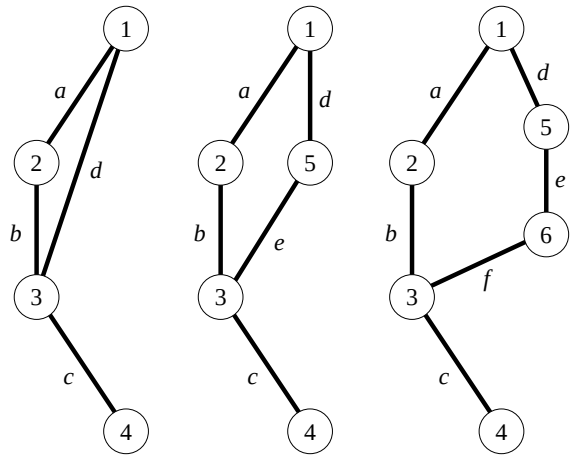


FIG. 2.4 – 3 exemples pour l'ajout d'un nouveau chemin entre deux nœuds existants

la figure 2.4 il ne sera plus possible d'accéder au nœud numéro 2. Souhaitant construire des graphes planaires dont tous les nœuds sont de degré au moins 3, cette configuration ne pourra pas satisfaire ce critère car le nœud 2 est de degré 2. Cette remarque implique une contrainte de construction très forte qui nous permet d'avoir assez peu de cas à traiter à chaque étape de l'algorithme : les chemins que nous construisons ne doivent pas «passer au dessus» d'un nœud qui ne soit pas de degré au moins 3.

Pour cette première étape, nous allons donc essayer de créer des chemins entre 1 et 2, ensuite entre 1 et 3 c'est impossible car le nœud 2 est de degré 2. C'est donc terminé avec comme départ le nœud 1. Pour le nœud 2 on peut seulement créer un chemin jusqu'au nœud 3. Enfin, pour le nœud 3, le seul chemin possible à partir de celui ci est en direction du nœud 4.

Les chemins que nous rajoutons correspondent, sur la décomposition, à l'ensemble des carrés situés entre deux plateaux horizontaux.

2.3.2 Suppression des redondances

Une seconde contrainte inhérente à tous les algorithmes d'énumération est d'empêcher la duplication des états déjà traités. Dans notre cas, si on ne traite pas ce problème, l'algorithme nous donne bien toutes les solutions de graphes planaires pour un nombre fixé d'arêtes mais il trouve aussi énormément de redondances dans les solutions, ce qui engendre une perte de temps importante.

La méthode que nous décrivons ici est plutôt intuitive et ce problème de redondances est assez gênant à résoudre. Un exemple, la figure 2.5, nous donne deux manières d'arriver au même état.

Soit on crée le chemin 1-5-2 et ensuite 5-3, soit on crée le chemin 2-5-3 et ensuite 1-5.

L'idée utilisée pour résoudre ce problème est telle qu'on ordonne la création des chemins, c'est-à-dire que l'on essaye de placer les chemins du haut vers le bas et une fois un chemin placé, on ne revient pas dessus. Prenons comme exemple la figure 2.5 : le premier schéma est correct mais pas le second car le chemin *f-d* pouvait être créé précédemment.

Si on prend notre graphe initial (figure 2.3), l'ordre des chemins à essayer est le suivant : $1 \to 2$, $1 \to 3$, $1 \to 4$, $2 \to 3$, $2 \to 4$, $3 \to 4$.

Cette technique est délicate car il faut bien faire attention à ne pas supprimer certaines solutions. En effet, empêcher la création de chemins qui pouvaient l'être dans une étape précédente est efficace mais il faut tenir compte des changements effectués depuis. Etudions les différents

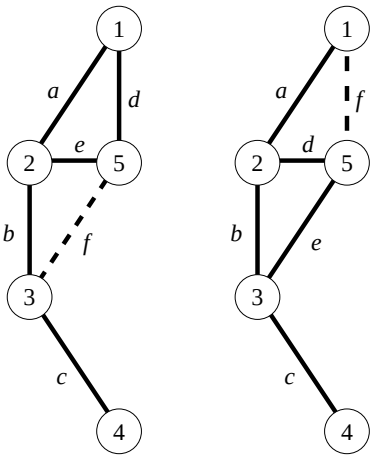


FIG. 2.5 – Exemple de redondance dans la construction de graphes planaires

cas qui se présentent lors du placement d'un nouveau chemin, pour cela examinons le cas de la figure 2.6.

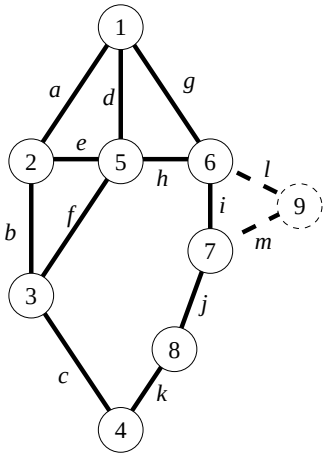


FIG. 2.6 – Exemple de construction de graphe planaire

Dans cet exemple, supposons que la dernière opération a été d'ajouter le chemin en pointillé contenant le nœud 9. Quels sont maintenant les chemins que nous avons le droit d'ajouter ?

1. De 1 à 6, non car nous pouvions le faire avant car ce chemin se trouve au dessus du dernier chemin placé.
2. De 1 à 9, non car de la même manière, nous pouvions le faire avant en créant un chemin contenant 1 nœud entre les nœuds 1 et 6.
3. De 1 à 7, oui nous pourrions le faire si le nœud 9 n'était pas de degré 2. De même pour les nœuds suivants.
4. De 6 à 9, oui car cette construction n'était pas réalisable avant.
5. De 6 à 7, oui mais interdit car le nœud 9 est de degré 2.
6. De 9 à 7, oui sans problème.
7. De 9 à 8, oui, il n'y a pas de problème car ce chemin est après le dernier construit.
8. De 9 à 4, oui mais interdit car le nœud 8 est de degré 2.

9. De 8 à 4, oui, il n'y a pas de problème.

Il est possible de voir qu'après le placement d'un chemin entre 2 nœuds i et j, il existe 3 zones de construction importantes qui sont :

Zone A : Les nœuds situés entre la source et i ;

Zone B : Les nœuds entre le suivant de i et le précédant de j ;

Zone C : Les nœuds entre j et le puits.

Ces différentes zones nous permettent de définir les chemins que nous pouvions placer auparavant.

- Dans la zone A il était possible de créer le chemin plus tôt.
- Entre A et B, il était possible de créer la même configuration (voir exemple figure 2.5).
- Tous les autres chemins doivent être traités car ils ne pouvaient pas être traités précédemment.

Cette méthode est rapide mais nous donne dans certains cas particuliers des solutions redondantes. Il faudra donc vérifier que des solutions n'aient pas été trouvées plusieurs fois.

2.3.3 Suppression des symétries

Les graphes étant la représentation de décompositions de rectangles en carrés, nous allons obtenir 8 fois les même solutions puisque notre problème a 8 symétries. Pour éviter de refaire les calculs sur des graphes qui vont donner des symétries de solutions déjà trouvées, nous avons mis au point un algorithme qui interdit de lancer la suite des calculs sur des graphes qui n'ont pas la «bonne» symétrie.

Une méthode consiste à mémoriser les graphes trouvés et à comparer tout nouveau graphe à toutes les symétries issues des graphes déjà trouvés. Cette méthode est impossible à mettre en œuvre car le nombre de graphes est bien trop important et le surcroît de calcul nécessaire est prohibitif.

Puisque notre algorithme d'énumération de graphe énumère toute les symétries, il suffit de choisir celle qui a une configuration particulière que n'ont pas les 7 autres. Nous nous sommes servi d'une idée proche de celle qui est définie dans code de Bouwkamp. En effet, pour qu'une solution soit unique, nous fixons des critères sur la position des carrés de plus grande taille. Dans notre cas, puisque nous n'avons pas encore les différentes tailles des carrés, nous avons choisi comme critère les degrés des nœuds de l'arbre.

Pour ce faire, le traitement des symétries sur les graphes n'est pas aussi simple que sur les carrés. En effet, sur les décompositions, trois opérations simples se combinent pour donner une des 8 symétries possibles : la symétrie verticale, la symétrie horizontale et la rotation de 90 degrés. Les opérations correspondantes sur les graphes sont : la symétrie verticale, la symétrie horizontale et le *graphe dual aux faces*. La figure 2.7 nous montre les 8 symétries possibles.

Nous allons obtenir dans notre énumération ces 8 configurations. Pour ne traiter qu'une seule d'entre elles, nous choisissons celle qui a le nœud de degré le plus élevé lorsque l'on parcourt le graphe de haut en bas et de gauche à droite. Dans notre exemple (figure 2.7) c'est le graphe ligne 1 colonne 3 qui sera accepté car dans l'ordre, le premier nœud a pour degré 3 et le deuxième a pour degré 4. Ainsi, lorsque notre algorithme trouve un graphe, nous le comparons à ses 7 symétries et si c'est celui qui satisfait nos critères alors on continue le traitement sur ce graphe, sinon on passe au graphe suivant.

Il n'est pas nécessaire de construire les différentes symétries. Une méthode plus rapide mais plus complexe à mettre en œuvre a été utilisée. Elle consiste à vérifier le degré de chaque nœud (ou de chaque face) sur le graphe initial mais en le parcourant de manière différente. Par exemple, plutôt que d'effectuer la symétrie verticale, il suffit de parcourir le graphe de bas

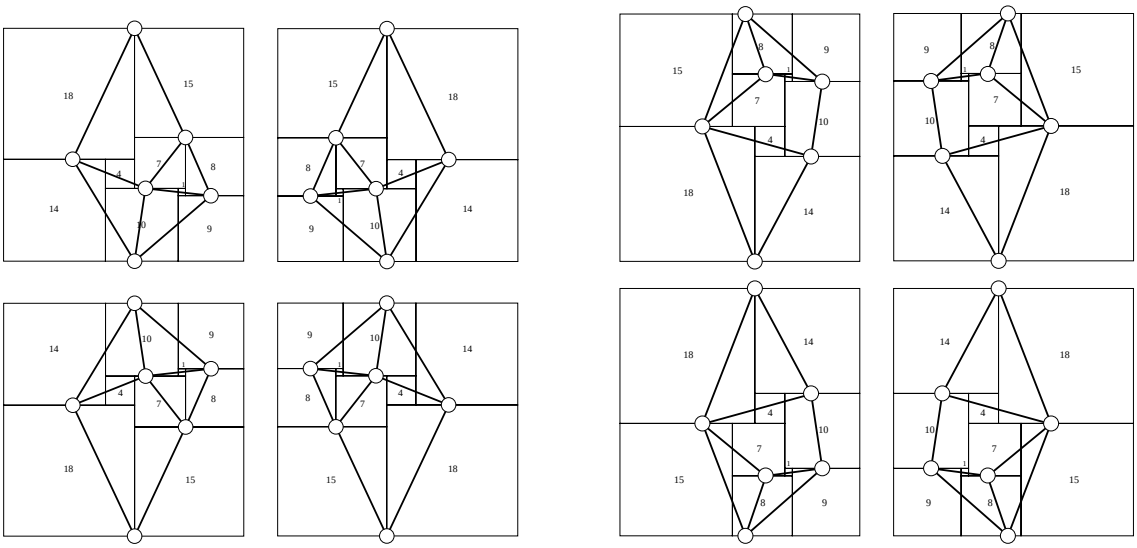


FIG. 2.7 – Les 8 symétries du problème

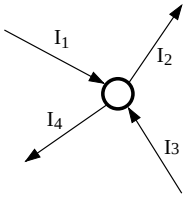
en haut et de gauche à droite plutôt que de haut en bas et de gauche à droite. Cette pratique est simple pour les symétries horizontales et verticales mais bien plus complexe pour le dual aux faces car il ne faut plus vérifier le degré des nœuds mais le degré des faces.

2.4 Construction des équations de Kirchhoff

Une fois que nous avons obtenu un graphe correct, il faut pouvoir calculer la taille des carrés. Sachant que ces tailles correspondent aux tensions sur les arêtes du graphe lorsque celles-ci ont une résistance unitaire, nous allons générer les équations nécessaires pour résoudre ce problème.

Pour cela, il existe les 2 lois de Kirchhoff qui nous permettent de trouver la valeur des différents courants.

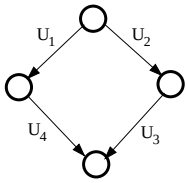
La *première loi de Kirchhoff* ou aussi *loi des nœuds*



La somme algébrique des intensités qui «arrivent» à un nœud est nulle. ($I_1 - I_2 + I_3 - I_4 = 0$)

On peut aussi écrire que : La somme des intensités qui arrivent au nœud est égale à la somme des intensités qui en repartent. ($I_1 + I_3 = I_2 + I_4$)

La *deuxième loi de Kirchhoff* ou aussi *loi des mailles*

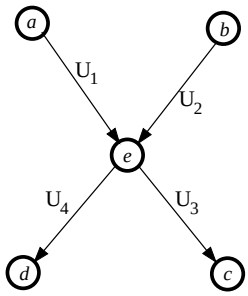


La somme algébrique des tensions rencontrées le long d'une maille est nulle. ($U_1 + U_4 - U_3 - U_2 = 0$)

Dans notre cas, puisque les tension sont égales aux intensités (les résistances sont unitaires), il est possible de tout exprimer en fonction des intensités.

Les équations générées ainsi grâce aux deux lois de Kirchhoff sont suffisantes pour que le système linéaire ainsi constitué soit soluble si on lui applique une tension à ses bornes. Donc, si nous avions un graphe comportant n arêtes, nous devons obtenir n équations linéairement indépendantes pour pouvoir trouver les n intensités $I_1, ..., I_n$. Le système à résoudre est donc un système $n \times n$ comportant uniquement des éléments de $\{-1, 0, 1\}$.

Il est possible de réduire la taille de ce système en calculant uniquement la tension en chacun des nœuds du graphe de la manière suivante :



Sur l'exemple, on a $U_1 = U_e - U_a$, $U_2 = U_e - U_b$, $U_3 = U_c - U_e$ et $U_4 = U_d - U_e$. Si maintenant on utilise la loi des nœuds sur cet exemple et puisque nous avons vu que la tension sur une arête est égale à son intensité, on obtient le résultat suivant :

$$U_1 + U_2 = U_3 + U_4$$

$$(U_e - U_a) + (U_e - U_b) = (U_c - U_e) + (U_d - U_e)$$

$$4 \times U_e - U_a - U_b - U_c - U_d = 0$$

L'avantage de cette modification est multiple :

1. L'orientation du graphe n'est plus nécessaire pour générer les équations ;
2. Le nombre d'équations et d'inconnues est égal au nombre de nœuds du graphe (inférieur au nombre d'arêtes) ;
3. La génération des équations est très simple à réaliser.

La matrice représentant le système à résoudre est finalement constitué par la matrice d'incidence (faite de 0 et de -1) avec sur la diagonale le degré de chacun des nœuds.

2.5 Résolution du système linéaire

Maintenant que nous avons réussi à fabriquer un système linéaire à coefficients entiers, pour trouver les différentes tailles de carrés, il ne nous reste plus qu'à le résoudre. Le système s'écrit donc de la façon suivante :

$$Ax = b$$

Nous connaissons la matrice A et le vecteur b qui contiennent des entiers et nous cherchons le vecteur des rationnels x.

Nous avons testé de nombreuses méthodes de résolution de systèmes linéaires à coefficients entiers. Nous avons fait une étude comparative de différents algorithmes en fonction des différents types de données pouvant être utilisés. Cette étude a été menée en collaboration avec David-Olivier Azulay qui travaille sur ces problèmes depuis longtemps [4, 5].

Voici les différents types de données que nous allons prendre en considération dans nos algorithmes :

les «long» sont des entiers sur 32 bits qui sont les entiers les plus grands gérés par notre processeur (de type Pentium) ;

les «longlong» sont des entiers sur 64 bits qui sont gérés par le compilateur (gcc) et non par le processeur, ce qui implique que les opérations effectuées sur ce type de variables peuvent être très coûteuses ;

les «float» sont des flottants sur 32 bits ayant une mantisse de 23 bits plus un bit de signe. La norme des flottants (IEEE 754) nous indique que tout résultat d'un calcul est le flottant le plus proche de la solution. Puisque nous travaillons sur les entiers, nos calculs, ayant des solutions entières ne dépassant pas les 23 bits, seront exacts ;

les «double» sont des flottants sur 64 bits ayant une mantisse de 52 bits plus un bit de signe. L'avantage de ce type de donnée est qu'il est géré directement par le processeur, il est ainsi possible d'utiliser ce type de donnée pour faire des calculs sur des entiers ne dépassants pas 52 bits.

2.5.1 Première approche

La première approche consiste tout simplement à utiliser la méthode de Gauss en faisant des divisions sur des nombres réels codés par des flottants. Cette méthode est efficace mais nous donne un résultat approché dans les réels alors que nous savons que les solutions sont rationnelles. De plus, pour pouvoir effectuer un dessin avec des nombres entiers, nous avons besoin de solutions rationnelles ayant un dénominateur commun le plus petit possible.

Nous avons donc modifié cette méthode de résolution pour effectuer les calculs sur des rationnels. D'une manière plus pratique, nous avons effectué tous les calculs sur des entiers en gardant en mémoire toutes les divisions à effectuer. Cette méthode est assez simple à mettre en œuvre mais pose le problème contraignant, celui de l'augmentation importante de la taille des nombres traités. Pourtant nos matrices sont assez creuses et ne contiennent que des -1 et des 0 sauf sur la diagonale où les nombres sont des entiers positifs plutôt faibles. Le tableau 2.1 nous donne, en fonction du nombre d'arêtes du graphe, le nombre maximum que génère cet algorithme durant les calculs.

Nb Arêtes	*Valeur Max*
9	10 125
10	132 000
11	6 853 632
12	42 998 169 600
13	46 438 023 168 000
14	2 779 523 414 900 711 424

TAB. 2.1 – Taille des nombres traités (sans division et sans PGCD)

Les types que nous avons défini précédemment ne nous permettent pas de dépasser 2^{63}. Au delà, il faut mettre en place des outils de gestion de nombres entiers de grandes tailles. Ces outils sont généralement très coûteux en terme de temps de calcul.

Par exemple, pour 14 arêtes, le calcul qui génère l'entier le plus important (2 779 523 414 900 711 424) est issu de la résolution du système suivant :

$$\begin{bmatrix} 5 & -1 & 0 & 0 & -1 & -1 & -1 \\ -1 & 4 & 0 & -1 & 0 & 0 & -1 \\ 0 & 0 & 3 & -1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 3 & -1 & 0 & 0 \\ -1 & 0 & 0 & -1 & 3 & 1 & 0 \\ -1 & 0 & 0 & 0 & -1 & 3 & -1 \\ -1 & -1 & 0 & 0 & 0 & -1 & 3 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Les nombres traités étant vraiment trop grands, nous allons étudier différentes méthodes en terme de temps de calcul et aussi en terme de taille des nombres générés.

2.5.2 Pivot sans division

La première idée est d'ajouter à l'algorithme précédent des calculs de PGCD pour diminuer la taille de ces nombres. Comme précédemment, nous n'effectuons aucune division lors du pivot mais par contre, nous divisons dès que nous le pouvons chaque ligne ayant un diviseur commun.

La méthode est décrite ci-dessous sous la forme d'un algorithme divisé en plusieurs morceaux.

Algorithme 1 $pivot\_sans\_division(a, x, b, n)$

for $l \leftarrow 1$ **to** n **do**
  CALL $recherche\_pivot\_non\_nul\_minimum$
  CALL $reduction\_ligne(l)$
  for $i \leftarrow l + 1$ **to** n **do if** $a[i, l] \neq 0$ **then**
    CALL $reduction\_ligne(i)$
    CALL $pivot$
CALL $calculer\_x$
// renvoie le dénominateur commun aux solutions
return d

Notre procédure $pivot\_sans\_division(a, x, b, n)$ résout le système $ax = b$ et reçoit les paramètres suivants :

- a est une matrice $n \times n$ d'entiers connus ;
- b est un vecteur d'entiers connus de taille n ;
- x est le vecteur de taille n qui va recevoir les solutions du système ;

Les solutions x pouvant être rationnelles, elles sont ramenées à un dénominateur commun renvoyé par la procédure.

Algorithme 2 $recherche\_pivot\_non\_nul\_minimum$

$piv \leftarrow l$
while $piv \leq n$ **and** $a[piv, l] = 0$ **do** $piv \leftarrow piv + 1$
if $piv > n$ **then** <*système insoluble*>
for $i \leftarrow piv + 1$ **to** n **do if** $a[i, l] \neq 0$ **and** $abs(a[i, l]) < abs(a[piv, l])$ **then** $piv \leftarrow i$
if $piv \neq l$ **then**
  <*échanger les lignes* piv *et* l *de la matrice* a>
  <*échanger* $b[piv]$ *et* $b[l]$>

La partie $recherche\_pivot\_non\_nul\_minimum$, comme son nom l'indique, recherche une ligne dans la matrice a pouvant servir de pivot. De plus, elle choisit la ligne ayant un pivot minimum (non nul) pour minimiser la taille des nombres générés.

Algorithme 3 $reduction\_ligne(i)$

$t \leftarrow b[i]$
for $j \leftarrow l$ **to** n **do** $t \leftarrow pgcd(t, a[i, j])$
if $t \neq 1$ **then**
  $b[i] \leftarrow b[i]/t$
  for $j \leftarrow l$ **to** n **do** $a[i, j] \leftarrow a[i, j]/t$

L'algorithme $reduction\_ligne(i)$ réduit la ligne numéro i en divisant chacun des éléments de la ligne par leur plus grand diviseur commun. Cette partie peut nous coûter cher en temps de calcul mais sans celle-ci, les nombres générés durant le calcul deviendrai trop grand.

Algorithme 4 $pivot$

$t \leftarrow pgcd(a[l, l], a[i, l])$
$p1 \leftarrow a[l, l]/t$
$p2 \leftarrow a[i, l]/t$
$b[i] \leftarrow p1 \times b[i] - p2 \times b[l]$
for $j \leftarrow l + 1$ **to** n **do** $a[i, j] \leftarrow p1 \times a[i, j] - p2 \times a[l, j]$

Dans l'algorithme $pivot$, on effectue simplement l'opération du pivot en prenant soin de calculer le plus grand diviseur commun des deux coefficients multiplicateurs, toujours pour minimiser la taille des nombres.

Algorithme 5 $calculer\_x$

$t \leftarrow pgcd(b[n], a[n, n])$
$d \leftarrow a[n, n]/t$
$x[n] = b[n]/t$
for $i \leftarrow n - 1$ **downto** 1 **do**
  $t \leftarrow 0$
  for $j \leftarrow n$ **downto** $i + 1$ **do** $t \leftarrow t + a[i, j] \times x[j]$
  $x[i] = b[i] \times d - t$
  $z \leftarrow pgcd(x[i], a[i, i])$
  $x[i] \leftarrow x[i]/z$
  $t \leftarrow a[i, i]/z$
  if $t \neq 1$ **then**
    $d = d \times t$
    for $j \leftarrow n$ **downto** $i + 1$ **do** $x[j] \leftarrow x[j] \times t$

Une fois la matrice a triangularisée, la dernière partie $calculer\_x$ permet de placer les solutions de notre système dans le vecteur x tout en créant le diviseur commun à ces solutions. De plus, cet algorithme nous donne le diviseur commun minimal.

Les résultats que nous apporte cet algorithme sont répertoriés dans le tableau 2.2.

Grâce au calcul du PGCD, les nombres traités sont vraiment plus petits qu'auparavant. Par exemple, pour le système linéaire précédent, le nombre le plus élevé généré avec cette méthode est 215 704 mais par contre, le calcul de nombreux PGCD peut être très coûteux.

Nb Arêtes	*Valeur Max*	*Temps (ms)*			
		longlong	*long*	*float*	*double*
9	660	0	0	0	0
10	2 255	0	0	0	0
11	6 549	0	0	0	0
12	27 648	20	10	10	0
13	94 348	80	30	30	30
14	356 481	300	150	140	120
15	1 290 459	1 220	590	510	460
16	4 879 681	5 090	2 310	2 100	1 800
17	16 281 225	19 940	9 520	8 800	7 300
18	62 362 609	82 230	38 730	36 270	29 500
19	178 377 486	333 680	156 780	146 030	122 510

TAB. 2.2 – Pivot sans division : Taille des nombres traités et temps

2.5.3 Pivot avec division exacte

Nous cherchons à présent à diminuer le nombre de calculs de PGCD tout en préservant l'efficacité apportée par l'algorithme quant à la taille des nombres générés. Cette nouvelle méthode que nous allons décrire provient de [32] qui montre qu'il est possible, à chaque itération, de diviser par le pivot de l'étape précédente.

Pour montrer comment cela est possible, prenons la matrice A suivante :

$$A^{(0)} = \begin{bmatrix} a & b & c & \dots & d \\ e & f & g & \dots & h \\ p & q & r & \dots & s \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

$A^{(0)}$ représente la matrice A à l'étape 0. Pour passer de l'étape $k-1$ à l'étape k, on applique la méthode d'élimination classique :

$$a_{ij}^{(k)} = a_{kk}^{(k-1)} a_{ij}^{(k-1)} - a_{kj}^{(k-1)} a_{ik}^{(k-1)} \quad i = k+1 \dots n,\ j = k+1 \dots n$$

Ainsi, on obtient la matrice suivante à l'étape 1 :

$$A^{(1)} = \begin{bmatrix} a & b & c & \dots & d \\ 0 & (af-be) & (ag-ce) & \dots & (ah-de) \\ 0 & (aq-bp) & (ar-cp) & \dots & (as-dp) \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots \end{bmatrix}$$

De même, à l'étape 2 :

$$A^{(2)} = \begin{bmatrix} a & b & c & \dots & d \\ 0 & (af-be) & (ag-ce) & \dots & (ah-de) \\ 0 & 0 & \begin{matrix}(a^2fr - a^2gq - aber \\ +abgp + aceq - acfp)\end{matrix} & \dots & \begin{matrix}(a^2fs - a^2qh - abes \\ +abhp + adeq - adfp)\end{matrix} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \dots & \dots \end{bmatrix}$$

On peut remarquer aisément que les lignes générées par la deuxième étape sont divisibles par a qui est le pivot de l'étape précédente. De même, les éléments modifiés dans $A^{(3)}$ seront divisibles par $a_{22}^{(1)} = af - be$. En réalité, à chaque itération cette division est possible.

Cette technique est pratique car elle permet de supprimer le calcul des PGCD. Pour être valide, cette technique nécessite d'effectuer le pivot sur toutes les lignes de la matrice, même celles qui commencent par un zéro, ce qui n'était pas le cas avant. Cette obligation, sachant que notre matrice est à 50% creuse, semble coûteuse, mais ce nouvel algorithme se simplifie quand même un peu lorsque la ligne commence par un zéro (voir l'algorithme 6).

Algorithme 6 $pivot\_avec\_division\_exacte(a, x, b, n)$

```
d ← 1
for l ← 1 to n do
  CALL recherche_pivot_non_nul_minimum
  for i ← l + 1 to n do
    if a[i, l] ≠ 0 then
      for j ← l + 1 to n do a[i, j] ← (a[l, l] × a[i, j] − a[i, l] × a[l, j])/d
      b[i] ← (a[l, l] × b[i] − a[i, j] × b[l])/d
    else
      for j ← l + 1 to n do a[i, j] ← (a[l, l] × a[i, j])/d
      b[i] ← (a[l, l] × b[i])/d
  d ← a[l, l]
// calcul des solutions
x[n] ← b[n]
for l ← n − 1 downto 1 do
  t ← b[l] × d
  for j ← l + 1 to n do t ← t − a[l, j] × x[j]
  x[l] ← t/a[l, l]
// renvoie le dénominateur commun aux solutions
return d
```

		Temps (ms)			
Nb Arêtes	*Valeur Max*	*long long*	*long*	*float*	*double*
9	750	0	0	0	0
10	2 565	0	0	0	0
11	8 704	0	0	0	0
12	34 056	10	0	0	0
13	104 168	30	20	10	10
14	357 210	150	60	40	30
15	1 070 764	640	280	190	190
16	3 378 483	2 900	1 180	620	570
17	11 914 525	12 220	4 510	2 150	2 330
18	36 119 461	51 260	18 970	9 280	9 540
19	110 977 552	219 900	79 820	37 650	39 310

TAB. 2.3 – Pivot avec division exacte : Taille des nombres traités et temps

2.5.4 Conclusion

Après les tests précédents, nous avons choisi l'algorithme du pivot avec division exacte car il est bien plus rapide que les autres méthodes et de plus, les nombres manipulés sont les plus petits.

Pour ce qui est du choix du type des variables utilisées, le type «float» ayant une précision trop faible, nous avons opté pour le type «double» qui nous fournit une précision suffisante sur les entiers (52 bits) avec des temps de calculs très intéressants.

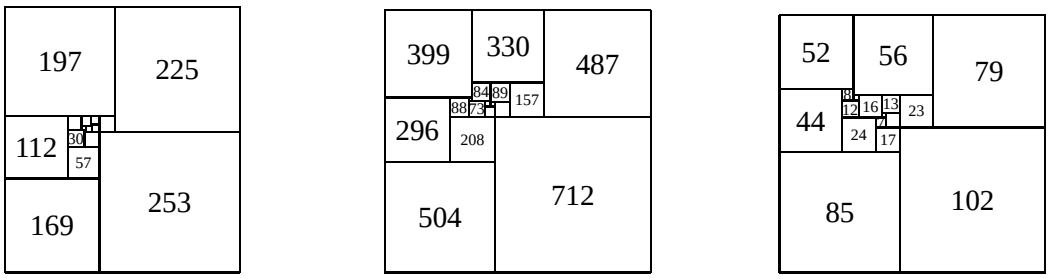
Par contre, ce choix nous a posé un problème car le temps d'affichage (lors du stockage des solutions dans un fichier ou de l'affichage à l'écran) d'un flottant est bien supérieur à celui d'un entier. En utilisant des variables flottantes pour l'affichage, nous passions autant de temps à afficher les résultats qu'à les calculer. Cette différence de temps d'affichage avec des variables de type entier est vraiment importante mais puisque nous nous intéressons uniquement à la partie entière des nombres flottants, il est beaucoup plus rapide de transformer les flottants en entiers avant de les afficher.

2.6 Résultats numériques

Avant de donner les résultats de cet algorithme, nous devons tout d'abord faire une remarque sur les solutions particulières que sont les décompositions croisées.

2.6.1 Cas des solutions croisées

L'algorithme nous donne des résultats satisfaisants mais nous trouvons plusieurs fois certaines solutions. Ces solutions redondantes sont en réalité les découpages croisés. Par exemple, les trois solutions de la figure 2.8 sont trouvées chacune deux fois.



422×478, 17 Carrés 1216×1199, 17 Carrés 187×181, 17 Carrés

FIG. 2.8 – Les 3 rectangles croisés d'ordre minimaux

Si on prend une solution et si on regarde le graphe qui l'a généré, on peut comprendre d'où vient cette redondance. En effet, si on prend pour exemple la solution $17 : 187 \times 181$, on obtient les deux graphes de la figure 2.9.

Ces graphes sont différents mais génèrent la même solution. Nous avons vu qu'un nœud du graphe correspond à un plateau horizontal. Ici, le plateau horizontal situé au dessus des carrés 17 et 102 peut être vu comme un seul plateau ou comme deux plateaux. Ainsi il est possible de générer deux graphes différents qui décrivent le même découpage (voir figure 2.9).

Ce problème n'est pas vraiment gênant car il existe peu de solutions croisées. Pour compter le nombre de solutions différentes, puisque nous trouvons chaque décomposition croisée 2 fois, et puisque ces décompositions sont facilement reconnaissables, il suffit d'enlever la moitié du nombre de solutions croisées au nombre de solutions total. Il faudra faire attention aux cas où il y a plus d'un croisement car chaque croisement multiplie par deux le nombre de solutions.

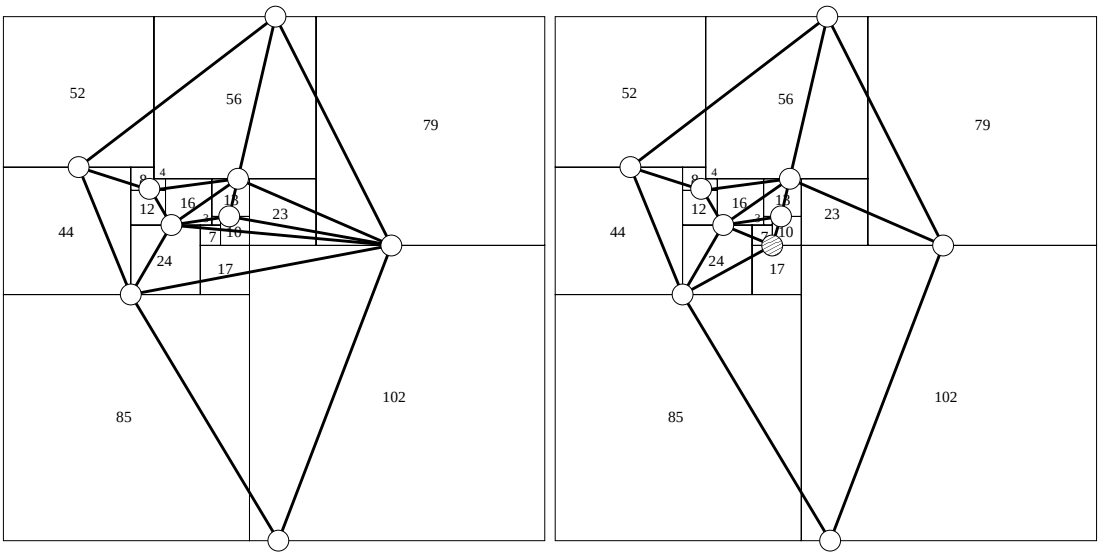


FIG. 2.9 – Graphes issus de la solution croisée 17 : 187×181

2.6.2 Résultats généraux

Le tableau 2.4 suivant nous donne l'ensemble des résultats obtenus avec l'algorithme précédent, bien sûr sans les symétries.

Les solutions de rectangles sont séparées en décompositions simples et composées, les chiffres entre parenthèses indiquent combien parmi les solutions sont croisées, et le point d'interrogation indique qu'il nous a été impossible de vérifier si certaines de ces solutions ont été trouvées en double. Par exemple le nombre de rectangles composés d'ordre 24 est de 87 844 807 dont 10 366 simplement croisées et 1 doublement croisée.

Le tableau 2.5 nous donne la répartition du temps de calcul suivant les différentes étapes de notre résolution. Au début de notre étude, nous passions la majorité du temps à résoudre des systèmes linéaires. Maintenant que nous avons analysé différentes approches de résolutions, c'est la partie qui énumère les graphes qui devient prépondérante.

2.6.3 Cas des carrés parfaits

Si on ne s'intéresse pas aux décompositions de rectangles, il est alors possible d'optimiser le programme pour la recherche de carrés parfaits. En effet, page 32, dans la propriété 6 nous avons prouvé que toute décomposition de carré ne peut avoir plus d'un côté avec 2 carrés, ce qui n'est pas vrai avec les rectangles. Ce test peut être effectué avant la résolution du système linéaire et donc diminuer le nombre de systèmes à résoudre. Dans ce cas, nous obtenons les résultats du tableau 2.6 suivant.

2.6.4 Cas des décompositions composées

La méthode que nous avons défini permet de trouver des décompositions simples et composées. Il est possible de chercher uniquement les décompositions simples en remarquant que les graphes correspondants aux décompositions composées contiennent un sous-graphe attaché au reste du graphe en deux points. Autrement dit, sur le graphe d'une décomposition simple, on ne peut pas rompre la connexité du graphe en supprimant deux de ses sommets. Un tel graphe est appelé 3-connexe.

Ordre	*Nombre de rectangles*		*Nombre de carrés*		*Nb d'arêtes placées*	*Nb de graphes générés*	*Nb de systèmes résolus*	*Valeur max durant la résolution*	*Temps*
	simples	*composés*	*simples*	*composés*					
9	2	0	0	0	387	120	5	750	0
10	6	0	0	0	1 214	433	14	2 565	0
11	22	0	0	0	3 991	1 554	46	8 704	10ms
12	67	0	0	0	13 461	5 561	148	34 056	50ms
13	213	4	0	0	46 375	20 023	512	104 168	140ms
14	744	48	0	0	161 900	72 270	1 757	357 210	500ms
15	2 609	264	0	0	571 933	262 339	6 238	1 070 764	1s 920ms
16	9 016	1 256	0	0	2 039 114	956 281	22 178	3 378 483	6s 480ms
17	(2) 31 426	(1) 5 396	0	0	7 330 129	3 502 757	80 022	11 914 525	23s 640ms
18	(2) 110 381	(3) 22 540	0	0	26 537 609	12 885 407	290 197	36 119 461	1m 26s
19	(7) 390 223	(11) 92 108	0	0	96 688 111	47 601 322	1 059 811	110 977 552	5m 14s
20	(29) 1 383 905	(58) 371 404	0	0	354 284 963	176 539 515	3 889 299	380 460 600	19m 08s
21	(166) 4 931 307 ?	(216) 1 472 300 ?	1	0	1 304 863 620	657 166 218	14 344 719	1 296 819 216	1h 16m
22	(444) 17 633 765 ?	(819) 5 781 252 ?	8	0	4 828 424 991	2 454 803 429	53 119 528	4 484 068 160	4h 54m
23	(1348) 63 301 415 ?	(2 996) 22 587 538 ?	12	0	17 943 113 602	9 199 713 216	197 488 605	15 545 475 072	19h 25m
24	(3892) 228 130 900 ?	(1) (10 366) 87 844 807 ?	26	4	66 940 459 597	34 582 819 551	736 775 227	45 030 384 826	3j 03h

TAB. 2.4 – Résultats de l'énumération de graphes planaires

Opération	*% du temps total de résolution* 16 arêtes	17 arêtes	18 arêtes	19 arêtes
Enumération des graphes	37,76 %	38,75 %	39,54 %	40,22 %
Test symétries dans graphes	5,61 %	5,63 %	5,78 %	5,91 %
Génération du système linéaire	8,84 %	10,03 %	11,06 %	12,02 %
Résolution du système linéaire	1,87 %	1,95 %	2,04 %	2,07 %
Calculs des tailles de carrés	18,37 %	17,42 %	16,51 %	15,91 %
Affichage des résultats	27,55 %	26,23 %	25,07 %	23,87 %

TAB. 2.5 – Répartition du temps de calcul en fonction des chaque étape de résolution

Ordre	*Nb de carrés* *simples*	*composés*	*Nb d'arêtes placées*	*Nb de graphes générés*	*Nb de systèmes résolus*	*Temps*
21	1	0	1 304 863 620	657 166 218	4 718 244	29m 27s
22	8	0	4 828 424 991	2 454 803 429	17 969 062	1h 51m
23	12	0	17 943 113 602	9 199 713 216	68 476 356	6h 47m
24	26	4	66 940 459 597	34 582 819 551	261 121 699	1j 02h
25	160	12	250 635 984 940	130 374 846 085	996 719 637	4j 05h
26	(1) 441	(1) 100	941 549 552 744	492 832 467 770	3 808 482 217	16j 05h

TAB. 2.6 – Résultats de l'énumération de graphes planaires pour des carrés parfaits

Nous n'avons pas appliqué cette modification à notre algorithme car nous souhaitons trouver toutes les solutions simples et composées.

Chapitre 3

Quelques propriétés des décompositions entières

Dans ce chapitre nous allons démontrer quelques propriétés sur les découpages en carrés distincts. Celles-ci sont essentielles pour l'obtention de nouveaux algorithmes fonctionnels que nous développerons dans la suite.

Tout d'abord, nous montrons qu'il s'agit d'un problème aux nombres entiers, c'est-à-dire que toute décomposition de rectangle peut s'écrire avec des tailles entières de carrés.

Nous donnons ensuite plusieurs propriétés assez simples mais très utiles sur la taille et sur l'emplacement de certains carrés.

Pour terminer, nous effectuons deux longues études de cas qui nous permettent de montrer que la plus petite taille de carré situé sur le bord d'une décomposition d'un rectangle en carrés distincts est 5 et la plus petite taille dans le coin est 9.

3.1 Passage à un problème aux nombres entiers

Nous allons montrer ici que nous traitons en fait un problème aux nombres entiers : toute solution de notre problème peut toujours s'exprimer avec des nombres entiers. Plus exactement nous allons prouver que les tailles des différents carrés sont toutes commensurables (dans des rapports rationnels).

Nous proposons deux démonstrations. La première, dont nous reprenons les grandes lignes, a été publiée par Max Dehn [20] en 1903. La deuxième, plus classique, utilise l'analogie du problème avec un réseau de résistances [2, 13, 37].

3.1.1 Démonstration de Max Dehn

Soit $D = \{(a_1, b_1), ..., (a_n, b_n)\}$ le découpage d'un rectangle $a_0 \times b_0$ en n rectangles $a_i \times b_i$. Nous allons montrer que si toutes les valeurs b_i/a_i, avec $i \neq 0$ sont rationnelles alors toutes les autres valeurs b_i/a_j, a_i/a_j, b_i/b_j sont rationnelles.

Bien entendu, on a :

$$a_0 b_0 = a_1 b_1 + a_2 b_2 + ... + a_n b_n \tag{3.1}$$

Soit maintenant S_x l'ensemble de toutes les équations linéaires homogènes à coefficients entiers admettant comme solutions $(x_0, ..., x_n) = (a_0, ..., a_n)$. De même appelons S_y l'ensemble de toutes les équations linéaires homogènes à coefficients entiers admettant comme solutions $(y_0, ..., y_n) = (b_0, ..., b_n)$:

$$S_x = \begin{cases} l_1^x(x_0, x_1, \ldots, x_n) = 0 \\ l_2^x(x_0, x_1, \ldots, x_n) = 0 \\ \vdots \end{cases} \qquad S_y = \begin{cases} l_1^y(y_0, y_1, \ldots, y_n) = 0 \\ l_2^y(y_0, y_1, \ldots, y_n) = 0 \\ \vdots \end{cases}$$

Par des considérations géométriques, Max Dehn montre alors l'implication suivante :

$$S_x \cup S_y \quad \Rightarrow \quad x_0 y_0 = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n \tag{3.2}$$

Supposons qu'il existe des nombres rationnels r_i tels que :

$$y_1 = r_1 x_1,\ y_2 = r_2 x_2, ...,\ y_n = r_n x_n \qquad (r_i > 0)$$

et considérons le système S obtenu en remplaçant dans $S_x \cup S_y$ les y_i par $r_i x_i$ $(i > 0)$

$$S = \begin{cases} l_1(x_0, y_0, x_1, \ldots, x_n) = 0 \\ l_2(x_0, y_0, x_1, \ldots, x_n) = 0 \\ \vdots \end{cases}$$

Du fait que les r_i sont rationnels, le système S ne fait intervenir que des coefficients rationnels.

De 3.2 on déduit que :

$$x_0 y_0 = r_1 x_1^2 + r_2 x_2^2 + ... + r_n x_n^2$$

Si maintenant on fixe $x_0 = -y_0$ on obtient :

$$x_0 = y_0 = x_1 = x_2 = ... = x_n = 0$$

Le système $S \cup \{x_0 = -y_0\}$, à $n+2$ variables, a donc une et une seule solution. Le nombre maximal de ses équations indépendantes est donc $n+2$. Le nombre maximal des équations linéairement indépendantes du système S, à $n+2$ variables, est donc $n+1$. Toute variable du système S peut donc s'exprimer comme une combinaison linéaire à coefficients rationnels des autres variables de S.

Ainsi on a montré que si les valeurs y_i/x_i $(i \neq 0)$ sont rationnelles, alors toutes les tailles $(x_0, ..., x_n, y_0, ..., y_n)$ sont dans des rapports rationnels. Ceci est vrai en particulier pour les $a_0, ..., a_n, b_0, ..., b_n$.

3.1.2 Utilisation d'un réseau électrique

La preuve la plus connue provient de l'analogie de notre problème avec un réseau de résistances unitaires (voir 2.2.1). En effet, nous avons vu précédemment que les tailles des carrés peuvent être trouvées en calculant les différences de potentiel se trouvant sur un réseau de résistances. Il est communément admis que la résolution d'un tel réseau est possible en utilisant les équations de Kirchhoff. Puisque ces équations sont linéaires à coefficients entiers, la solution est donc rationnelle.

Cette démonstration est très rapide mais repose sur le fait qu'il a été démontré que les équations de Kirchhoff donnent une et une seule solution. La plupart des livres traitant des équations de Kirchhoff citent cette propriété sans la démontrer.

Voici une démonstration de l'existence et de l'unicité de la solution des équations de Kirchhoff. Elle nous a été donnée par Jean-François Maurras lors d'une réunion d'équipe :

Soit $G = (X, U)$ un graphe fini orienté et connexe. Appelons $S(X, U)$ sa matrice d'incidence sommet arc :

$$\forall u \in U \; u = (x, y) \quad S(x, u) = -1 \quad S(y, u) = 1 \quad \forall z \notin \{x, y\} \; S(z, u) = 0$$

A chaque arc u est associé une résistance r_u ($= 1$ ici). Appelons $I(X)$ un vecteur de courants injectés aux sommets. Bien entendu, on a $\sum_{x \in X} I(x) = 0$. De façon symétrique, appelons $V(X)$ un vecteur potentiel défini en chaque sommet. Appelons $R(U, U)$ la matrice telle que $\forall u \neq v, R(u, v) = 0$ et $R(u, u) = r_u$, et soit $A(U, U)$ son inverse (e.g. $A(u, u) = \frac{1}{r_u}$). Enfin appelons $i(U)$ un vecteur de courants indicés par les arcs du graphe G et $\theta(U)$ un vecteur de tensions indicées, lui aussi par les arcs. On a :

$$\begin{cases} S(X,U)i(U) = I(X) & loi\,des\,noeuds, \\ \theta(U) = S^t(X,U)V^t(X) & loi\,des\,mailles\ ({}^t\text{: }transition) \\ \theta(U) = R(U,U)i(U) & loi\,d'Ohm \end{cases}$$

Des deux dernières équations on tire :

$$i(U) = A(U, U)S^t(X, U)V^t(X)$$

Avec la première on a :

$$S(X, U)A(U, U)S^t(X, U)V^t(X) = I(X) \tag{3.3}$$

Tout serait pour «le mieux dans le meilleur des mondes» si la matrice qui prémultiplie le vecteur $V^t(X)$ dans l'expression précédente était inversible. Mais il y a peu de chance qu'elle le soit car un potentiel est défini à une constante près.

Soit $X' = X \setminus \{y\}$, et soit $M(X', X') = S(X', U)A(U, U)S^t(X', U)$. Montrons dans notre contexte que $M(X', X')$ est inversible. Etant donné la définition de $A(U, U)$ ici, la matrice identité, on a :

$$M(X', X') = S(X', U)S^t(X', U)$$

Supposons donc que l'on ait une combinaison linéaire nulle à coefficients $V(X') \neq 0(X')$ des colonnes de $M(X', X')$. Que signifie une combinaison linéaire nulle des colonnes de cette matrice $M(X', X')$?

$$\sum_{x \in X'} M(X', x)V(x) = 0(X')$$

En écriture matricielle :

$$M(X', X')V(X') = 0(X')$$

En revenant à la définition de $M(X', X')$:

$$S(X', U)S^t(X', U)V(X') = 0(X')$$

Par associativité d'une part, le vecteur $V(X') \neq 0(X')$ prolongé en le vecteur $V(X)$ par la valeur $V(x) = 0$ étant interprété comme un potentiel sur le graphe initial $G(X, U)$ d'autre part, le produit :

$$S^t(X', U)V(X') = S^t(X, U)V(X) = \theta(U)$$

une tension **non-nulle** sur $G = (X, U)$.
Un vecteur $\phi(U)$ est, par définition, un flot sur $G = (X, U)$, si :

$$S(X, U)\phi(U) = 0(X)$$

L'équation au sommet y, $S(y, U)\phi(U) = 0(y)$ se déduit des équations à tous les autres sommets, car par définition de la matrice $S(X, U)$, on a :

$$S(y, U) = -\sum_{x \in X \setminus \{y\}} S(x, U)$$

Donc si $S(X', U)\phi(U) = 0(X')$, alors $S(X, U)\phi(U) = 0(X)$, et donc $\phi(U)$ est un flot.
D'autre part, $\theta(U)$ étant une tension et $\phi(U)$ un flot, on a $\theta^t(U)\phi(U) = 0$, en d'autre termes tout flot est orthogonal à toute tension, en effet :

$$(V^t(X)S(X, U))\phi(U) = V^t(X)(S(X, U)\phi(U)) = V^t(X)0(X) = 0$$

Revenons à notre problème. On a vu, en revenant à la définition de $M(X', X')$, qu'une combinaison linéaire nulle de coefficients $V(X')$ des colonnes de cette matrice pouvait être interprétée comme le produit nul de $S(X', U)$ par la tension $\theta(U)$ non-nulle. Ce produit étant nul, ce vecteur non-nul $\theta(U)$ ($\neq 0(U)$) est aussi un flot. Il est donc orthogonal à lui même, ce qui est impossible dans les espaces vectoriels sur le corps $\mathbb{Q}$ des rationnels (de caractéristique nulle, c'est, en revanche possible pour les corps finis...), une contradiction.

Le même raisonnement s'applique si $A(U, U)$ est une matrice diagonale dont les éléments de la diagonale sont tous strictement non-nuls.

La matrice $M(X', X')$ à coefficients rationnels est donc inversible et son inverse est à coefficients rationnels. Choisissons un vecteur de courants injectés $I(X)$ correspondant à notre problème, c'est-à-dire, si les sommets correspondants aux sommets supérieur et inférieur du grand carré sont respectivement x et y :

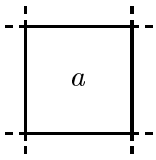
$$I(x) = 1 \quad I(y) = -1 \quad \forall z \in X \setminus \{x, y\}\ I(z) = 0$$

Le vecteur $V(X')$ déduit au moyen de l'égalité (3.3) est donc unique et rationnel. Ses valeurs sont les ordonnées, rationnelles, des autres carrés.

3.2 Prolongements obligatoires

Nous utiliserons le terme de *plateau* pour désigner un segment de taille maximale dans une décomposition d'un rectangle en carrés. Il s'agit en réalité d'un segment de droite formé par un ou plusieurs côtés de carrés de la décomposition. Par exemple, on peut trouver un plateau horizontal dans la figure 1.1 (page 2) formé par la partie supérieure des carrés de tailles 4 et 37.

Nous utiliserons le terme de *prolongement* pour désigner un segment partant du coin d'un carré et prolongeant un de ses côtés. Cette définition permet de faire apparaître une propriété inhérente à tout découpage en carré : dans une décomposition tous les coins d'un carré sont soit prolongés horizontalement, soit verticalement ou quelquefois des deux manières, ceci exception faite des coins du rectangle décomposé.



Par exemple le coin en haut à gauche est forcément prolongé soit vers le haut, soit vers sa gauche soit dans certains cas prolongé dans les deux directions (c'est le cas pour le carré de taille 102 dans la figure 3.1). Dans ce dernier cas, on a une décomposition croisée. Plus précisément, une décomposition est dite *croisée* si 4 coins de carrés se rejoignent en un point.

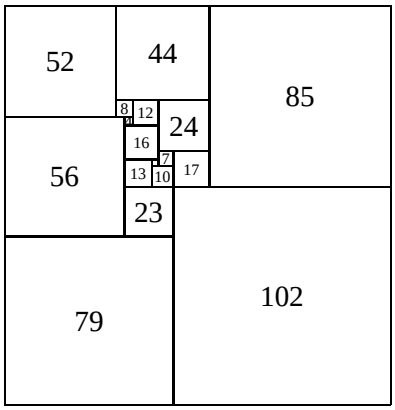


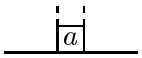
FIG. 3.1 – Rectangle croisé de taille 181×187, 17 carrés

Rappelons l'évidence qui nous sera bien utile dans la partie 5 :

Propriété 1 *Tout carré élément d'une décomposition est forcement prolongé en chacun de ses coins, exception faite des coins de la décomposition.*

3.3 Taille minimale dans une décomposition entière

Les carrés de tailles 1 et 2 sont des carrés assez particuliers car ils ne peuvent pas être décomposés en somme d'entiers différents. Cette propriété les empêche de se trouver au milieu d'un plateau, on les trouve donc seulement au bord. Pour le prouver, supposons que le carré a soit au milieu d'un plateau. Il ne peut être prolongé vers la droite car a est non décomposable en somme d'entiers distincts. C'est aussi le cas pour le prolongement à gauche. Donc les deux coins supérieurs se prolongent vers le haut, ce qui est aussi impossible pour les mêmes raisons.



Ainsi pour ces carrés non décomposables, ils se trouvent forcément dans une des 2 configurations suivantes :

Propriété 2 *Les carrés de tailles 1 et 2 ne peuvent se trouver au milieu d'un plateau.*

Cette propriété est intéressante pour limiter l'énumération lors de la construction d'une solution. Par contre, lorsque l'on détient une solution, cette propriété est aussi vraie pour son plus petit carré.

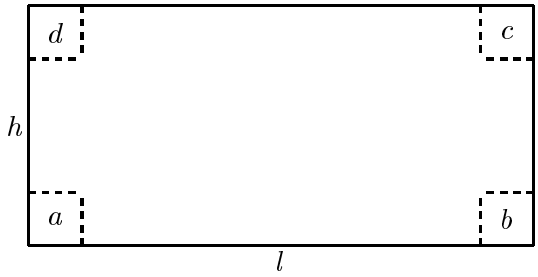
Propriété 3 *Le plus petit carré d'une décomposition ne peut se trouver au milieu d'un plateau.*

Le plus petit carré dans une décomposition est forcément placé dans une des deux configurations précisées précédemment. De même, le plus petit carré d'une décomposition ne peut se trouver sur le bord.

Propriété 4 *Le plus petit carré d'une décomposition ne peut se trouver au bord.*

3.4 Nombre de carrés au bord d'une décomposition parfaite

On veut connaître les différentes propriétés concernant le nombre de carrés au bord des décompositions parfaites. Pour cela, on part du rectangle suivant :



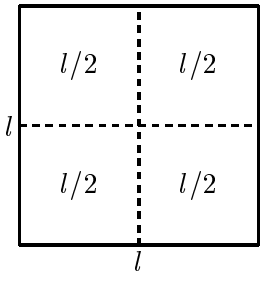
On veut montrer ici qu'un des 2 côtés les plus larges est constitué par strictement plus de 2 carrés.

Tout d'abord on peut facilement se convaincre que les cotés les plus longs ne peuvent être constitués par un seul carré. Supposons maintenant que les 2 cotés les plus longs soient constitués par exactement 2 carrés. Nous avons donc les carrés de tailles a et b en bas de la figure (donc on a $a + b = l$) et les carrés de tailles c et d en haut (donc $c + d = l$). De plus, sur le côté gauche se trouvent au moins les deux carrés a et d (donc $a + d \leq h$) et de même à droite ($b + c \leq h$).

On a donc les équations suivantes :

$$\left.\begin{array}{r} h \leq l \\ a+d \leq h \\ b+c \leq h \end{array}\right\} \Rightarrow \left.\begin{array}{c} a+d \leq l \\ b+c \leq l \end{array}\right\} \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} \Rightarrow a=b=c=d=\frac{l}{2} \Rightarrow a^2+b^2+c^2+d^2=l^2$$
$$\left.\begin{array}{r} a+b=l \\ c+d=l \end{array}\right\} \Rightarrow a+b+c+d=2l$$

On peut donc voir ici que la seule décomposition de rectangle ayant sur ses côtés les plus larges exactement 2 carrés est un carré décomposé en exactement 4 carrés égaux.



Ce carré n'étant pas parfait, on obtient la propriété suivante :

Propriété 5 *Dans toute décomposition parfaite d'un rectangle en carrés entiers, aux moins 3 carrés se trouvent sur un des côtés de taille maximale du rectangle.*

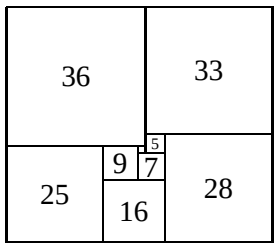


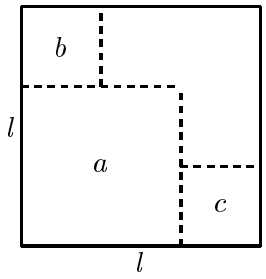
FIG. 3.2 – Décomposition parfaite de rectangle avec le minimum de carrés sur les bords (69×61 d'ordre 9)

Par contre, il est possible d'avoir 2 carrés sur les 2 côtés les moins larges (cf. figure 3.2).

Dans le cas où un des côtés les plus courts est constitué par un seul coté de carré, il est simple de voir que la propriété 5 est toujours vérifiée.

3.5 Nombre de carrés au bord d'un carré parfait

Nous allons nous intéresser ici au cas particulier des carrés. D'après la propriété 5, il est impossible d'avoir un carré parfait ayant 2 carrés sur des bords opposés. Par contre, peut-on avoir deux carrés sur des bords adjacents ?



Une remarque sur cette figure, c'est que le carré de taille a ne peut pas être plus petit que b ou c. De plus on a les équations suivantes :

$$\left.\begin{array}{l} a+b=l \\ a+c=l \end{array}\right\} \Rightarrow\ b=c=l-a$$

Ici, les carrés b et c étant de même taille, il nous sera impossible d'obtenir une décomposition parfaite. Ainsi nous avons la propriété suivante :

Propriété 6 *Dans une décomposition parfaite de carré, il y a au plus 1 côté constitué par 2 côtés de carrés.*

Par contre on ne peut rien dire pour 3 carrés aux bords (cf. figure 3.3).

3.6 Taille minimale au bord d'une décomposition entière

Les algorithmes que nous allons décrire plus loin construisent la solution en partant de configurations de carrés déjà placés sur un des côté de la surface à décomposer.

La connaissance de la plus petite taille d'un carré sur le bord d'une décomposition en carrés entiers a un grand impact sur le temps de résolution, surtout lorsqu'il s'agit d'énumérer l'ensemble des configurations possibles. Nous allons donc chercher à réduire cette énumération de

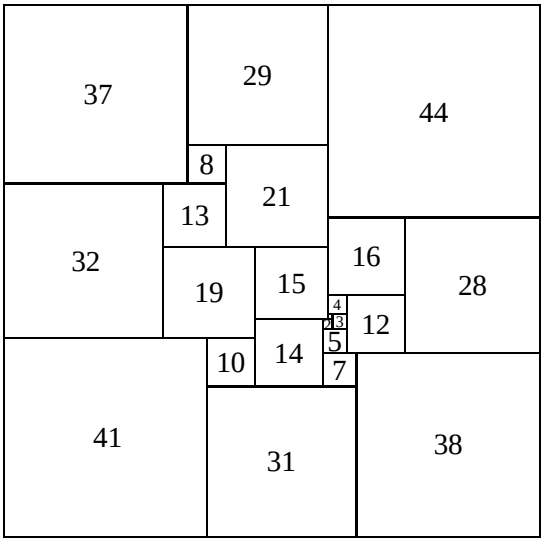


FIG. 3.3 – Carré parfait 110×110 d'ordre 22

configurations possibles pouvant se trouver sur le bord d'une décomposition de carré en carrés distincts. Ce problème revient à décomposer la taille du bord du carré en somme d'entiers tous différents. Par exemple, pour un carré de taille 6 nous obtenons les 11 configurations suivante : (6), (5,1), (4,2), (3,2,1), (3,1,2), (2,4), (2,3,1),(2,1,3), (1,5), (1,3,2), (1,2,3).

La valeur des tailles que nous souhaitons décomposer est de 112 pour pouvoir espérer trouver la solution des 21 carrés (figure 1.1). Un problème se pose ici car le nombre de décompositions trouvées par un algorithme d'énumération total est bien trop élevé, même si l'on pense à enlever la symétrie du problème. Pour se donner un ordre d'idée sur les grandeurs mises en jeu, il suffit de regarder le tableau 3.1 qui donne un aperçu du résultat lorsqu'on effectue une énumération totale.

Taille	*Configurations*	*Temps (ms)*
6	11	0
40	751 491	850
50	10 759 321	13 650
60	134 264 861	183 950
70	1 488 522 537	2 148 990
80	14 627 858 515	22 390 120

TAB. 3.1 – Décomposition d'un entier en somme d'entiers différents

Grâce à ce tableau, il est possible de voir qu'il sera certainement difficile d'arriver ou même seulement de traiter une taille de 112, surtout qu'il ne s'agit ici que de placer les carrés se trouvant sur un côté de la décomposition. Même si de nombreuses configurations vont être rapidement rejetées par l'algorithme de recherche, il faudra quand même les générer. Ainsi, pour pouvoir obtenir des temps raisonnables, il faut essayer d'enlever au plus tôt les configurations ne pouvant pas donner de solutions.

Nous allons montrer ici quelles sont les tailles minimales que peuvent avoir les carrés au bord d'une décomposition en carrés entiers. Le tableau 3.2 nous donne une idée du gain que nous pouvons obtenir en fonction de la taille du plus petit carré possible au bord. Ce tableau se lit de la façon suivante. Si on désire énumérer le bord d'un carré de taille 40, l'algorithme d'énumération total va nous donner 751 491 solutions (d'après le tableau 3.1) qui se répartissent

en 636 494 solutions ayant pour carré le plus petit un carré de taille 1, 91 058 solutions ayant pour carré le plus petit un carré de taille 2, et ainsi de suite.

	Taille à décomposer					
Plus petite valeur	6	40	50	60	70	80
1	8	636 494	9 235 316	116 961 146	1 316 005 544	13 051 521 446
2	2	91 058	1 246 256	14 430 398	146 280 428	1 355 360 546
3	0	17 180	211 250	2 252 408	21 210 950	181 519 340
4	0	4 448	46 670	460 340	3 820 178	31 026 920
5	0	1 370	13 088	110 750	863 180	6 215 258
6	1	518	3 908	31 826	228 128	1 538 918
7	0	224	1 526	10 436	71 474	443 816
8	0	92	650	4 112	24 902	140 660
9	0	38	308	1 706	9 248	52 358
10	0	26	152	782	4 340	20 882

TAB. 3.2 – Répartition des décompositions d'entiers en fonction de leur plus petit élément

Pour que ce tableau soit plus clair, nous avons placé la répartition des configurations pour un carré de taille 6. Si on regarde les configurations possibles pour cette valeur, 8 ont pour valeur minimale 1, 2 ont pour valeur minimale 2 et 1 seule configuration a pour valeur minimale 6, ce qui nous fait bien un total de 11 configurations.

L'intérêt de ce tableau est de montrer que le nombre de décompositions possibles avec un carré de taille 1 est très important, les solutions représentent approximativement 90% de l'ensemble des décompositions. De même, si on enlève les décompositions ayant un carré de taille 1, alors l'ensemble des décompositions restantes avec un carré de taille 2 représente approximativement 85% de l'ensemble des décompositions. C'est pour cela que la connaissance du plus petit carré au bord d'une décomposition nous permettra de gagner vraiment beaucoup de temps de calcul.

Nous allons étudier différentes tailles de carrés sur le bord d'une décomposition. Il ne faut pas oublier que nous nous sommes placés dans le cadre où toutes les tailles de carrés de la décomposition sont des entiers. Pour alléger les démonstrations, nous n'avons pas explicité certaines configurations qui, trivialement, ne peuvent mener à une solution.

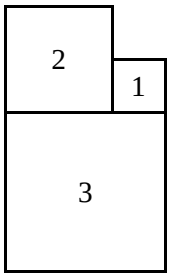
3.6.1 Carrés de tailles 1 et 2

Le carré de taille 1 ne peut se trouver sur le bord d'une décomposition pour une raison simple. Supposons que ce carré soit sur le bord. Il sera forcement entouré de carrés plus grands et donc le seul carré que l'on puisse placer au dessus du carré 1 est un autre carré de taille 1, ce qui est impossible puisque tous les carrés ont une taille différente.

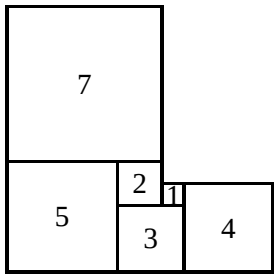
Pour un carré de taille 2, la démonstration est exactement la même puisque 2 non plus n'est pas décomposable en une somme d'entiers différents.

3.6.2 Carré de taille 3

Maintenant que nous avons montré que les carrés de tailles 1 et 2 ne sont pas sur le bord, le carré de taille 3 est forcément entouré de carrés plus grands. La seule décomposition possible étant $3 = 2 + 1$, nous obtenons forcément la configuration suivante (ou le symétrique) :



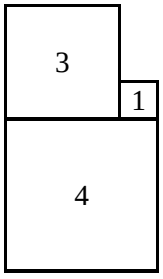
1. Le carré 1 ne peut être prolongé vers le haut puisque 1 est non décomposable. Il est donc forcément prolongé vers la droite par le carré de taille 4.
2. Le carré que nous pouvons placer au dessus des carrés de taille 1 et 4 est forcément plus grand que 1 donc le carré 2 ne peut être prolongé sur la droite. 2 étant non décomposable, le carré de taille 2 doit forcément se prolonger sur la gauche par le carré de taille 5.
3. Le plateau de longueur 7 ainsi obtenu ne peut être prolongé ni à droite, ni à gauche, il faut donc le prolonger verticalement avec le carré de taille 7.



A partir de cette étape, nous ne pouvons pas prolonger le carré de taille 4 sur la droite, il faut donc prolonger verticalement le plateau de longueur 5. Ceci étant impossible, nous obtenons un échec. Il est donc impossible d'avoir un carré de taille 3 au bord d'une décomposition en carrés entiers.

3.6.3 Carré de taille 4

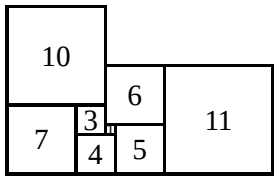
Le carré de taille 4 pose un peu plus de problèmes car l'arbre de recherche est assez grand. Tout d'abord, le carré de taille 4 est forcement entouré de carrés plus grands, donc il faut le décomposer par les carrés de tailles 3 et 1. Ce qui nous donne pour commencer, la configuration suivante :



On remarque que :

1. Le carré de taille 1 étant non décomposable, il est forcement prolongé à droite par un carré de taille 5.
2. Le plateau de longueur 6 ainsi obtenu n'est pas prolongeable à droite. 6 étant non décomposable, on est obligé de placer le carré de taille 6 au dessus.

3. Le plateau de longueur 3 n'est pas prolongeable verticalement, il faut donc le prolonger sur la gauche par le carré de taille 7.
4. Nous obtenons un plateau de longueur 10 non prolongeable à gauche et non décomposable. Nous sommes donc obligés de placer le carré de taille 10 dessus.
5. Le plateau de taille 6 n'est pas décomposable et non prolongeable verticalement, on doit donc le prolonger sur la droite par le carré de taille 11.

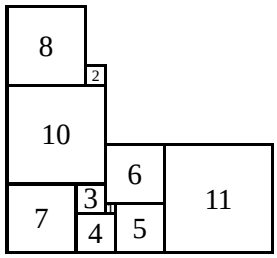


Jusqu'ici, nous n'avons pas eu de choix sur les carrés placés mais à partir de cette configuration plusieurs choix vont être possibles. Nous allons donc travailler sur le carré qui nous donne le moins de cas à traiter, le carré de taille 10. Il est facile de remarquer que ce carré n'est pas prolongeable sur la droite. Donc deux choix s'offrent à nous. Soit le carré de taille 10 est prolongé verticalement, soit il l'est sur sa gauche.

Cas 1 - Prolongement vertical du carré de taille 10

Nous allons supposer ici que le carré de taille 10 n'est pas prolongeable sur sa gauche. Il faut donc le prolonger verticalement et la seule décomposition possible est $10 = 8 + 2$ ou $10 = 2 + 8$.

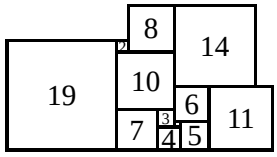
Si on décompose le carré de taille 10 en $8 + 2$ on obtient cette configuration :



Ce cas nous mène à une impasse puisque 2 n'est pas décomposable (donc non prolongeable verticalement) et le carré de taille 2 n'est pas prolongeable à droite par un carré de taille 8 ou par une de ses décomposition. Nous devons donc décomposer le carré de taille 10 en $2 + 8$.

A partir de là on peut faire les remarques suivantes :

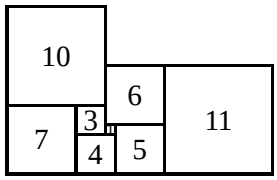
1. Le carré de taille 2 n'étant pas décomposable, nous sommes obligés de le prolonger sur la gauche par un carré de taille 19.
2. Le carré de taille 8 n'est pas décomposable et ne peut être prolongé sur la gauche par un carré de taille 6 ou une de ses décompositions. Le carré de taille 8 est donc prolongé sur la droite par un carré de taille 14.



Le carré de taille 11 n'est pas prolongeable à droite. Nous avons donc un plateau de largeur 3 à décomposer. Ceci étant impossible, nous en déduisons que le carré de taille 10 n'était pas prolongeable verticalement.

Cas 2 - Prolongement à gauche du carré de taille 10

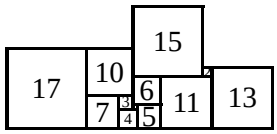
Nous revenons donc à la configuration suivante :



Le carré de taille 10 peut être prolongé à gauche de trois manières. Soit par un carré de taille 17 soit par la somme des carrés 2 et 15 soit par les carrés 8 et 9.

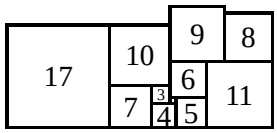
Sous-cas 2.1 - Prolongement à gauche par un carré de taille 17 Prolongeons le carré de taille 10 à gauche par un carré de taille 17. Le carré de taille 11 n'étant pas prolongeable à droite, il faut donc décomposer le plateau de taille 17 verticalement. Encore une fois, deux solutions s'offrent à nous. Soit on place deux carrés de tailles 15 et 2, soit on place les deux carrés 9 et 8.

Sous-cas 2.1.1 - Décomposition avec les carrés de taille 15 et 2 De manière évidente, il est impossible de placer le carré de taille 2 à gauche de celui de taille 15. Nous plaçons donc le carré de taille 2 à droite. Ce carré de taille 2 n'étant pas décomposable, nous sommes obligés de le prolonger à droite par le carré de taille 13.



Le carré de taille 13 n'est pas prolongeable sur sa droite. Il faut donc décomposer verticalement le plateau de longueur 15, ce qui est impossible. Il faut donc revenir en arrière.

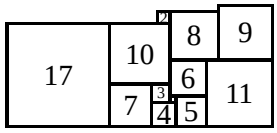
Sous-cas 2.1.2 - Décomposition avec les carrés de tailles 9 et 8 Si nous plaçons le carré 9 à gauche du carré de taille 8, nous obtenons ceci :



Ici, le carré de taille 9 ne peut être prolongé ni à droite, ni à gauche, et 9 n'est pas décomposable. Nous obtenons donc un échec.

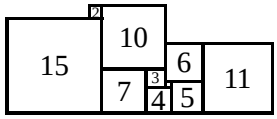
Il faut donc placer le carré de taille 8 à gauche de celui de taille 9.

Le carré 8 n'étant pas décomposable, il faut donc le prolonger à gauche par un carré de taille 2, ce qui nous donne ceci :



Le carré de taille 2 n'étant pas prolongeable à gauche, il faut donc décomposer le plateau de longueur 10. Ceci étant impossible, nous obtenons donc encore une fois un échec.

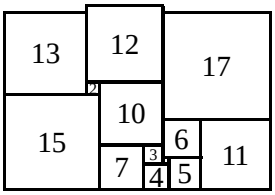
Sous-cas 2.2 - Prolongement à gauche par les carrés de taille 2 et 15 Nous ne pouvons pas placer le carré de taille 2 en dessous de celui de taille 15 sinon nous obtenons trivialement un échec. Nous avons donc la configuration suivante :

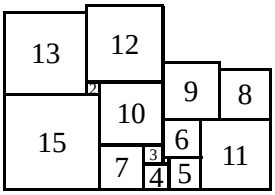


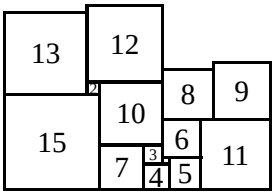
On remarque que :

1. Le carré de taille 15 n'étant pas prolongeable à gauche, nous devons placer le carré de taille 13 au dessus.
2. Le carré de taille 10 n'étant pas prolongeable à droite, nous sommes obligés de placer le carré de taille 12 au dessus.

Le carré de taille 11 n'étant pas prolongeable à droite, nous avons les trois choix suivants pour prolonger verticalement le plateau de longueur 17 :

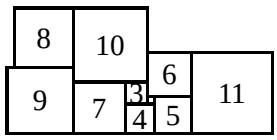






– Dans le premier cas, le carré de taille 12 ne sera prolongeable ni à droite, ni à gauche. Comme il est aussi impossible de le décomposer, nous aboutissons à un échec.
– En plaçant le carré de taille 9 à gauche de celui de taille 8, le carré de taille 9 ne sera pas prolongeable à droite et ne peut être prolongé verticalement.
– En plaçant les carrés 8 puis 9, le carré de taille 8 n'étant pas prolongeable verticalement nous aboutissons aussi à un échec.

Sous-cas 2.3 - Prolongement à gauche par les carrés de taille 8 et 9 De manière évidente, nous ne pouvons placer le carré de taille 8 en dessous de celui de taille 9. On place donc le carré de taille 8 au dessus et nous obtenons la configuration suivante :



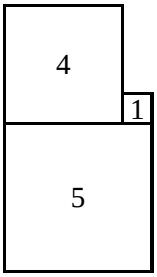
Le carré de taille 9 ne peut être prolongé sur sa gauche. Nous devons donc décomposer le plateau de largeur 1, ce qui est impossible donc nous obtenons notre dernier échec.

Il ne peut donc y avoir de carré de taille 4 sur le côté d'une décomposition en carrés entiers.

3.6.4 Carré de taille 5

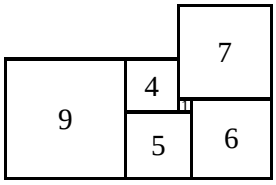
Comme dans toutes les démonstration précédentes, puisqu'il n'existe pas de carré de taille inférieure à 5 au bord d'une décomposition, notre carré de taille 5 ne peut être prolongé ni à droite, ni à gauche. Il faut donc le prolonger verticalement. Dans les parties précédentes, nous avions au plus un seul choix pour la première décomposition. Ici, nous avons deux cas. Soit 5 est décomposé en 4 et 1, soit en 3 et 2.

Cas 1 - Décomposition en $\{4, 1\}$



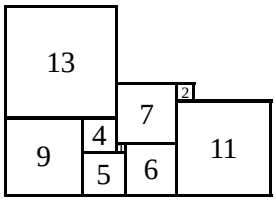
On remarque que :

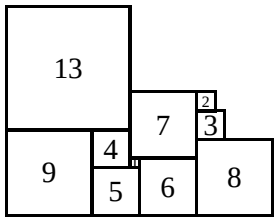
1. Le carré de taille 1 est non prolongeable verticalement, on doit donc le prolonger sur la droite par le carré de taille 6.
2. Le carré de taille 6 n'étant pas prolongeable à droite, on est obligé de placer le carré de taille 7 au dessus.
3. Enfin, le carré de taille 4 étant non décomposable, il faut le prolonger sur la gauche par le carré de taille 9.

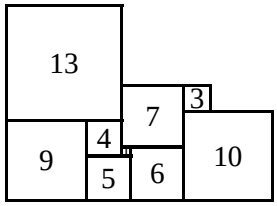


Le carré de taille 9 n'est pas prolongeable à gauche, il faut donc prolonger verticalement le plateau de taille 13. Ceci peut se faire de 4 façons différentes.

Sous-cas 1.1 - Carré de taille 13 Plaçons un carré de taille 13 sur le plateau de longueur 13. Le carré de taille 7 étant non décomposable, il faut donc le prolonger sur la droite. Pour cela, trois solutions que voici :



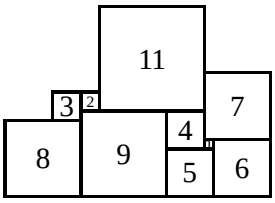




– En décomposant avec les carrés 2 et 11, nous sommes obligés de mettre le carré de taille 11 au dessous. A partir de cette configuration, le carré de taille 2 n'est pas prolongeable sur la droite. Il faut donc décomposer le plateau de taille 9. Ceci est impossible donc cette configuration n'est pas la bonne.
– En décomposant 13 avec les carrés 2, 3 et 8, nous sommes obligés de les placer par ordre croissant de taille du haut vers le bas comme sur la figure. Dans ce cas, et pour les mêmes raisons que précédemment, le carré de taille 2 n'est pas prolongeable sur la droite et il est impossible de remplir le plateau qui se trouve au dessus. On obtient donc aussi un échec.
– Si on décompose 13 avec les carrés 3 et 10, nous sommes obligés de placer le carré de taille 3 au dessus du carré de taille 10. A ce moment la, un problème se pose. Le carré de taille 3 n'est pas prolongeable à droite, il faut donc prolonger verticalement le plateau de taille 10. De plus, le plateau de taille 7 se trouvant sur le carré de taille 10 ne peut être

prolongé verticalement, il faut donc prolonger le carré de taille 10 vers la droite. Nous avons donc besoin de 2 décompositions pour la valeur 10 alors que la seule solution est $8 + 2$. Il nous est donc impossible de continuer la construction.

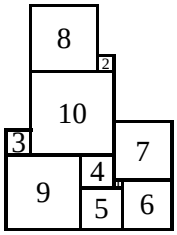
Sous-cas 1.2 - Carrés de tailles 11 et 2 Si nous plaçons les carrés de tailles 11 et 2, nous devons placer le carré de taille 2 à gauche. 2 n'étant pas décomposable, il faut le prolonger sur la gauche par les carrés 8 et 3, le carré de taille 3 devant être placé au dessus du carré de taille 8.

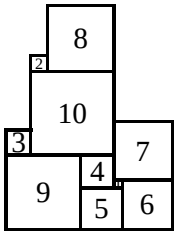


Le carré de taille 3 ne pouvant être prolongé sur la gauche, nous devons combler le plateau de largeur 5. Ceci étant impossible, nous obtenons un échec.

Sous-cas 1.3 - Carrés de tailles 10 et 3 Ici, le carré de taille 3 doit être placé à gauche du carré de taille 10 pour éviter un échec.

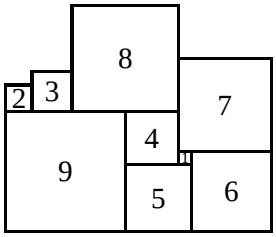
Ce carré de taille 10 ne peut être prolongé ni sur la droite, ni sur la gauche, il faut donc prolonger verticalement le plateau de largeur 10. La seule solution est d'utiliser les carrés 2 et 8.





Dans ces deux cas, le carré de taille 2 n'est pas prolongeable. On obtient donc un échec.

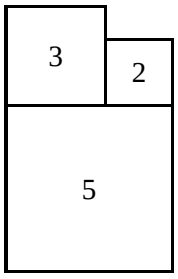
Sous-cas 1.4 - Carrés de tailles 8 et 3 et 2 Pour l'emplacement de ces trois carrés, nous devons les placer dans l'ordre croissant de gauche à droite pour éviter un échec trivial. On a donc la configuration suivante :



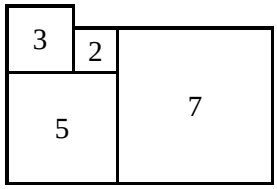
Ici, le carré de taille 3 n'est pas prolongeable. On obtient donc encore un échec.

Cas 2 - Décomposition en $\{3, 2\}$

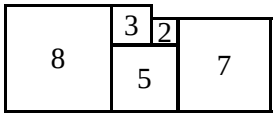
Nous avons élagué une partie de l'arbre et maintenant nous sommes sûr que s'il y a une solution avec un carré de taille 5 au bord, alors nous aurons forcément la configuration suivante :

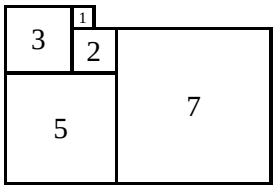


Le carré de taille 2 n'étant pas décomposable, nous devons prolonger à droite par le carré de taille 7.



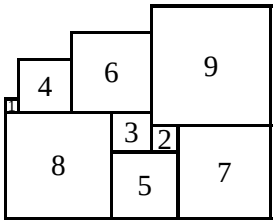
A partir de cette figure, nous avons plusieurs choix possibles pour continuer la construction. Si un carré de taille 5 peut se trouver au bord d'une décomposition alors nous savons que cette dernière configuration sera forcément utilisée. Regardons maintenant les différents choix possibles. Le carré de taille 3 ne peut être prolongé verticalement, il est donc forcément prolongé horizontalement à gauche ou à droite, ce qui nous donne les choix suivants :

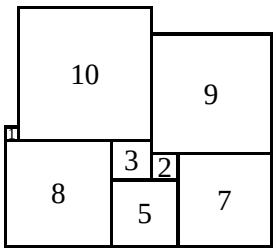


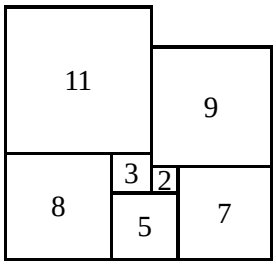


Sous-cas 2.1 - Prolongement du carré 3 à gauche Dans le cas ou l'on prolonge le carré de taille 3 à gauche avec un carré de taille 8, le plateau de taille 9 n'étant pas prolongeable à droite, on est forcé de le prolonger verticalement avec un carré de taille 9.

De même, le plateau de taille 11 n'étant pas prolongeable à gauche il faut le prolonger verticalement. Ici, le nombre de choix pour décomposer le plateau de taille 11 est plus important. On peut soit prolonger par les carrés de tailles 1, 4 et 6, soit utiliser les carrés 1 et 10, soit tout simplement placer le carré de taille 11.

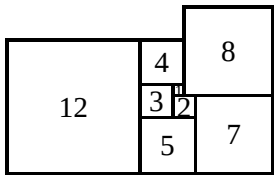




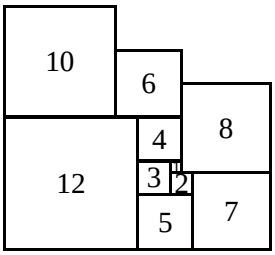


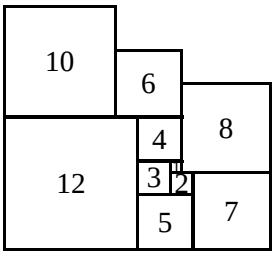
Dans les deux premiers cas, il n'est pas possible de prolonger le carré de taille 1 verticalement, il faut donc le prolonger à gauche. Ce prolongement est impossible puisque tous les carrés de 1 à 9 sont utilisés. Il ne nous reste donc que le dernier cas. Le nombre de choix possibles à cet instant sont assez importants et ne mènent pas trivialement à une impasse. Il est difficile de continuer cette démonstration plus loin dans cette direction. Pour le moment, gardons cette configuration pour plus tard.

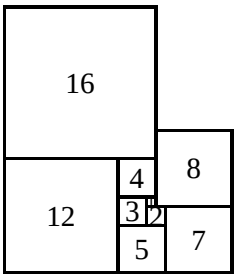
Sous-cas 2.2 - Prolongement du carré 3 à droite Si nous prolongeons le carré de taille 3 à droite avec un carré de taille 1, nous obtenons alors un plateau de taille 8 qui n'est pas prolongeable horizontalement. On place donc le carré de taille 8 mais dans ce cas, le plateau de taille 4 n'est plus prolongeable horizontalement, il nous faut donc le prolonger verticalement avec le carré de taille 4. Ce carré de taille 4 n'étant pas prolongeable verticalement, on le prolonge à gauche par le carré de taille 12. On obtient ainsi la configuration suivante :



Cette configuration nous offre plusieurs choix pour le plateau de taille 16. Celui-ci ne pouvant être prolongé à gauche, on peut le prolonger verticalement de trois façons, avec les carrés de tailles 10 et 6 ($10 + 6 = 6 + 10$) ou avec le carré de taille 16.



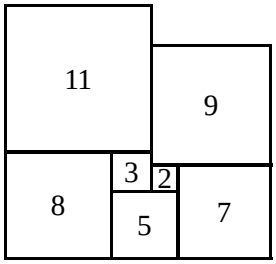


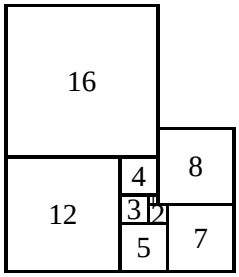


Dans le premier cas, le carré de taille 10 ne peut être ni prolongé horizontalement, ni verticalement. Dans le deuxième cas, c'est le carré de taille 6 qui ne peut être prolongé. Il reste donc la dernière configuration.

A partir de cette configuration, le nombre de choix explose considérablement et il est impossible de développer ici la suite de la construction.

Sous-cas 2.3 - Conclusion Nous avons donc obtenu deux configurations possibles. Si le carré de taille 5 peut se trouver au bord d'une décomposition en carrés entiers nous aurons alors forcément une de ces deux configurations.





Il est impossible d'aller plus loin de façon manuelle. En partant du principe utilisé ici pour faire nos démonstrations, il a été possible de développer un programme permettant d'aller beaucoup plus loin. Malheureusement, ce programme n'a jamais pu aboutir ni en prouvant que le carré de taille 5 ne pouvait se trouver au bord, ni en trouvant une solution possible avec le carré de taille 5 au bord.

3.6.5 Conclusion

Nous avons fini par trouver dans nos résultats deux carrés ayant un carré de taille 5 au bord de la décomposition entière.

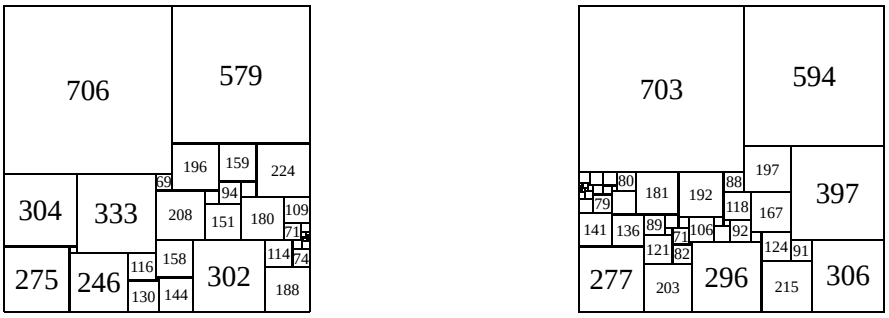


FIG. 3.4 – Deux décompositions ayant un carré de taille 5 au bord

Ces deux carrés (figure 3.4) s'écrivent de la façon suivante en utilisant le code de Bouwkamp :

- (706,579) (196,159,224) (304,333,69) (94,65) (208,57) (180,109) (151) (71,38) (23,15) (8,7) (2,**5**) (10,16,4,1) (3) (12) (158,302,114,40,6) (34) (275,29) (74) (246,116) (188) (14,144) (130)
- (703,594) (197,397) (47,56,59,80,181,192,88) (15,23,9) (24,41) (38,21) (7,8) (**5**,2) (1,4,16,10) (3) (12) (34) (101) (28) (118,167) (79) (62) (141) (136,89,57) (46,106,40) (26,92) (66) (32,71) (43,124) (121) (91,306) (11,296) (82) (277) (215) (203)

Nous avons montré ici que les carrés ayant une taille allant de 1 à 4 ne pouvaient pas se trouver sur le bord d'une décomposition en carrés entiers, par contre, pour le carré de taille 5, nous savons qu'il existe des solutions.

Propriété 7 *Dans une décomposition en carrés de tailles entières, aucun carré de taille 1 à 4 ne peut se trouver sur le bord.*

Il faut bien voir que les gains obtenus (tableau 3.3) portent uniquement sur l'énumération des carrés se trouvant sur un bord. Les configurations que nous enlevons grâce à cette démonstration ne donnent pas un gain final aussi important car les configurations supprimées auraient très vite bloqué lors de leurs utilisations dans la recherche d'une décomposition. Par exemple, si on essaye de placer un carré de taille 1 sur le bord, l'algorithme de construction aura du mal à aller très loin dans sa recherche.

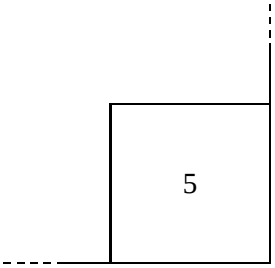
3.7 Taille minimale au coin d'une décomposition entière

Nous allons montrer ici que certaines tailles de carrés sont impossibles dans le coin d'une décomposition en carrés entiers. Nous procéderons de la même manière que précédemment pour la taille minimale d'un carré au bord. Sachant qu'il n'existe pas de carré de taille inférieure à 5 sur le bord, ceci implique qu'il n'existe pas de carré inférieur à 5 dans les coins non plus. Nous commencerons donc notre recherche avec le carré de taille 5.

	Enumération Totale		*Tailles > 4*		*Gain (Rapport)*	
Taille	*Configurations*	*Temps (ms)*	*Configurations*	*Temps (ms)*	*Configurations*	*Temps*
40	751 491	850	2 311	0	325	-
50	10 759 321	13 650	19 829	10	542	-
60	134 264 861	183 950	160 569	90	836	2 043
70	1 488 522 537	2 148 990	1 205 437	780	1 234	2 755
80	14 627 858 515	22 390 120	8 430 263	6 080	1 735	3 682
90			57 302 415	43 430		
100			363 094 951	300 350		
110			2 231 998 829	1 930 100		
120						

TAB. 3.3 – Gain obtenu sur l'énumération initiale en sachant que le plus petit carré a une taille de 5.

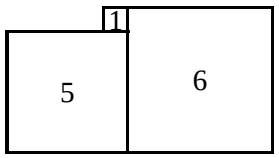
3.7.1 Carré de taille 5



Pour le carré de taille 5, il doit forcément être prolongé soit à gauche, soit verticalement, ce qui revient au même par symétrie. Il faut donc décomposer le carré de taille 5 avec d'autres carrés plus petits mais on a vu qu'il n'existe aucune décomposition avec un carré de taille inférieure à 5 au bord. Donc ce prolongement du carré de taille 5 est impossible, donc le carré de taille 5 ne peut se trouver dans un coin.

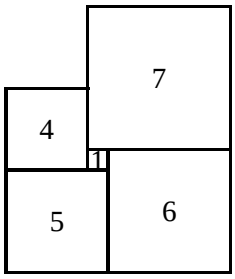
3.7.2 Carré de taille 6

Pour le carré de taille 6, le seul moyen de le décomposer est d'utiliser le carré de taille 5 au bord.



On remarque que :

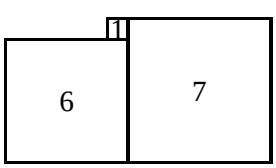
1. Le carré de taille 5 ne peut être prolongé sur la gauche, il faut donc prolonger le plateau de largeur 4 verticalement avec le carré de taille 4.
2. Le plateau de taille 7 restant ne peut être prolongé que par le carré de taille 7.

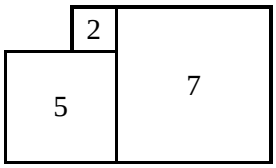


Le carré de taille 7 ne peut pas être prolongé à gauche et ne peut pas non plus être prolongé verticalement donc le carré de taille 6 ne peut se trouver dans un coin.

3.7.3 Carré de taille 7

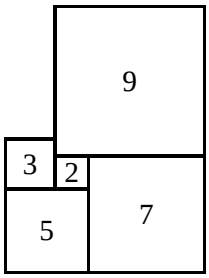
Le carré de taille 7 peut être prolongé de deux façons :

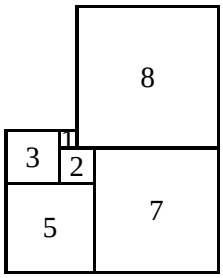




Cas 1 - Décomposition en $\{5, 2\}$

Après avoir utilisé les carrés de tailles 5 et 2, on va s'intéresser au prolongement du carré de taille 5. Si on essaye de prolonger ce carré sur la gauche, on ne peut qu'utiliser les carrés 4 et 1 et dans ce cas, le carré 1 ne sera pas prolongeable. Il faut donc prolonger verticalement le carré de taille 5 par un carré de taille 3. Dans cette configuration, prolongeons le plateau de largeur 9. Il existe seulement deux façons de le faire :



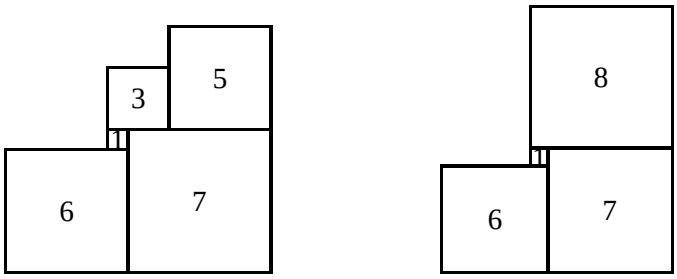


- Dans le premier cas, le carré de taille 3 ne peut pas être prolongé verticalement, il faut donc le prolonger horizontalement avec le carré de taille 8. A ce moment là, le carré de taille 9 devient non prolongeable.
- Dans le deuxième cas de figure, le carré de taille 8 est non prolongeable.

On tombe ainsi dans une impasse il faut donc essayer une autre décomposition pour le carré de taille 7.

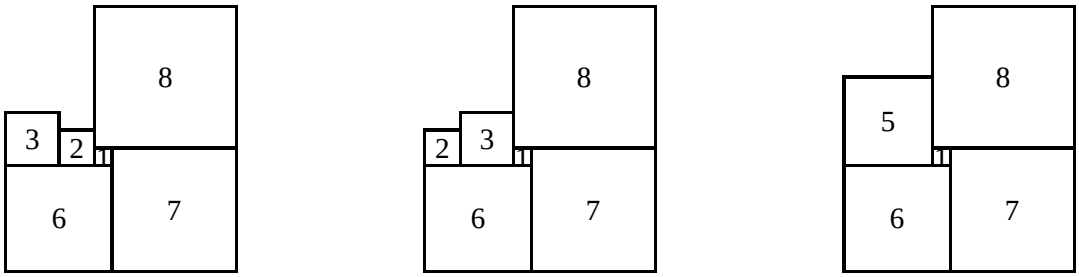
Cas 2 - Décomposition en $\{6, 1\}$

Le plateau de taille 8 ne peut pas être prolongé sur la gauche il doit donc être prolongé verticalement. Il peut être prolongé soit avec les carrés 5 et 3, soit avec le carré de taille 8.

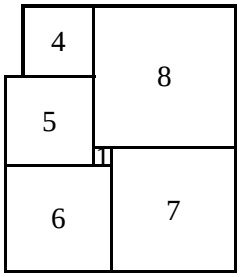


Dans le premier cas, l'utilisation des carrés 3 et 5 pose un problème. Le carré de taille 3 ne peut être prolongé verticalement, il doit donc être prolongé sur la gauche par le carré de taille 4. On obtient ainsi un plateau de largeur 1 sur le carré de taille 6 qui ne peut pas être prolongé. Ce n'est donc pas le bon choix.

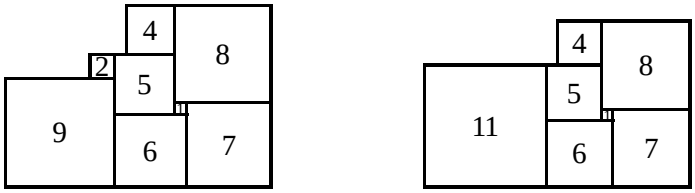
Le deuxième cas nous laisse encore plusieurs choix pour le prolongement vertical du plateau de largeur 5 puisqu'il ne peut pas être prolongé horizontalement.



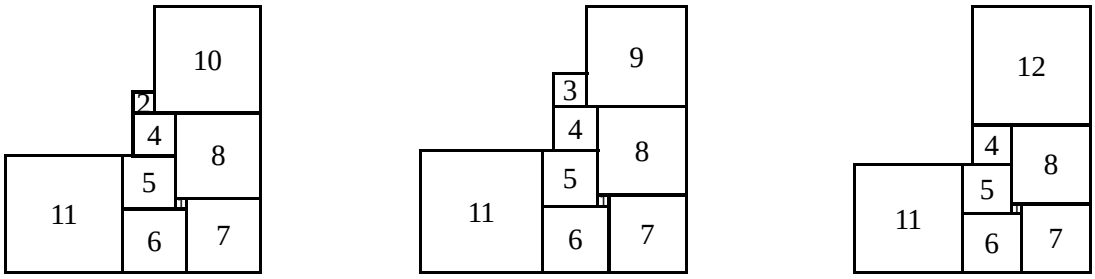
- La décomposition en $3+2$ est impossible car on ne peut pas prolonger le carré de taille 2.
- La décomposition en $2+3$ est aussi impossible car le carré de taille 3 n'est pas prolongeable.
- Cette dernière configuration est donc la seule possible pour pouvoir obtenir un carré de taille 7 dans le coin. Ici, le carré de taille 8 n'étant pas prolongeable verticalement doit l'être horizontalement par le carré de taille 4.



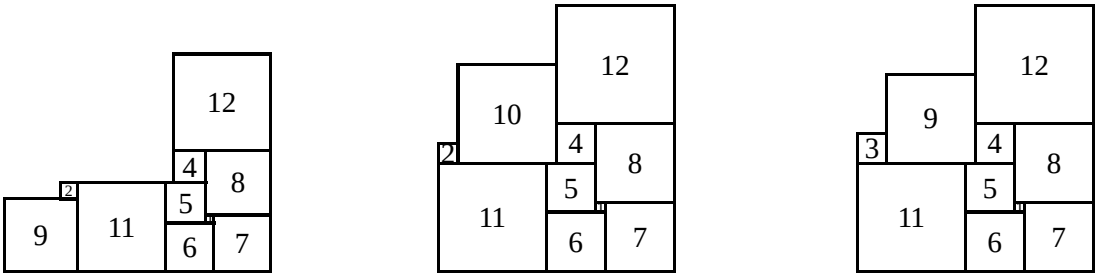
Le plateau de largeur 1 se trouvant sur le carré de taille 5 n'étant pas prolongeable verticalement doit se prolonger sur la gauche avec, soit les carrés 9 et 2, soit le carré de taille 11.



- Dans le premier cas, le carré de taille 9 n'est pas prolongeable sur la gauche et le plateau de largeur 7 se trouvant au dessus, ne peut être prolongé verticalement.
- Dans la deuxième configuration, le carré de taille 4 ne peut être prolongé sur sa gauche, il faut donc prolonger le plateau de largeur 12 verticalement avec soit les carrés 2 et 10, soit les carrés 3 et 9 ou soit le carré de taille 12.



- Dans le premier cas, le carré de taille 2 n'est prolongeable ni horizontalement ni verticalement.
- Dans la deuxième configuration, c'est le carré de taille 3 qui n'est pas prolongeable.
- On se retrouve donc avec la troisième configuration comme seule solution possible. Le carré de taille 11 peut être prolongé soit horizontalement avec les carrés 9 et 2, soit verticalement avec les carrés 10 et 2 ou les carrés 9 et 3.



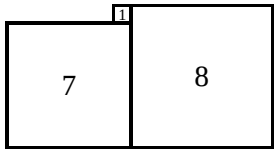
- Dans la première configuration, le carré de taille 9 ne peut être prolongé.
- Dans cette deuxième configuration le carré de taille 10 et le carré de taille 12 sont non prolongeables.
- Enfin, dans cette dernière configuration, c'est le carré de taille 9 qui est non prolongeable.

Nous avons fini d'explorer toutes les configurations possibles pouvant avoir un carré de taille 7 dans un coin. Toutes ont mené à un échec ce qui nous permet de dire qu'il est impossible de fabriquer une décomposition avec des carrés entiers ayant un carré de taille 7 dans le coin.

3.7.4 Carré de taille 8

Pour le carré de taille 8, il existe 4 prolongement possibles qui utilisent les carrés $\{5, 2, 1\}$, $\{5, 3\}$, $\{6, 2\}$ ou $\{7, 1\}$.

Cas 1 - Décomposition en $\{7, 1\}$

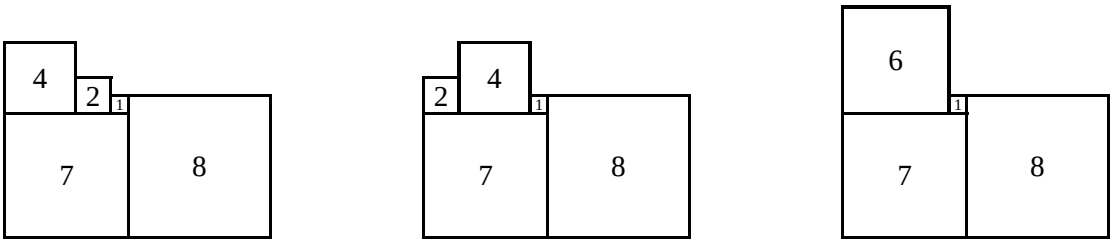


Sous-cas 1.1 - Prolongement horizontal du carré 7 Pour effectuer un prolongement horizontal du carré de taille 7, il existe 2 solutions principales :

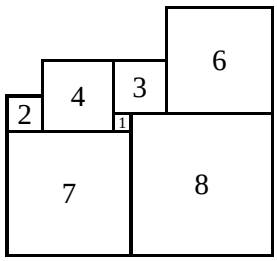
– Dans le premier cas, le plateau de largeur 8 sur le carré de taille 7 n'est prolongeable ni horizontalement ni verticalement.
– Dans la deuxième configuration c'est le plateau de largeur 1 sur le carré de taille 4 qui n'est pas prolongeable.

Il est donc impossible de prolonger horizontalement le carré de taille 7.

Sous-cas 1.2 - Prolongement vertical du carré 7 Ce prolongement peut s'effectuer de 3 façons :



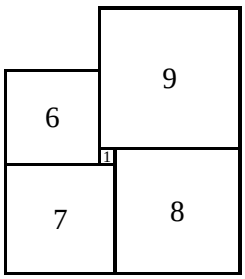
– Dans le premier cas, le carré de taille 2 ne peut être prolongé ni horizontalement, ni verticalement.
– Dans la deuxième configuration, le seul prolongement possible pour le carré de taille 4 est un prolongement à droite avec un carré de taille 3, ce qui génère un plateau de largeur 6 sur le carré de taille 8. La seule façon de prolonger ce plateau est de placer le carré de taille 6.



Ici, le carré de taille 6 n'est pas prolongeable, il ne nous reste donc plus que le troisième cas à traiter.

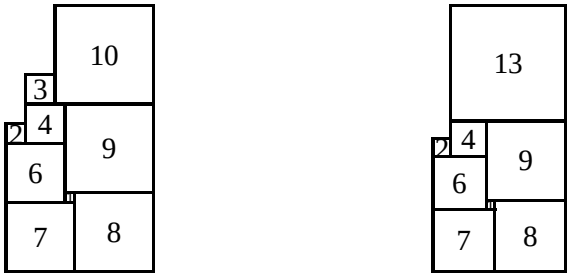
Dans la troisième configuration, le plateau de largeur 9 peut être comblé de nombreuses façons avec les différents ensembles de carrés suivants : $\{9\}$, $\{5,4\}$, $\{4,3,2\}$.

– Si on utilise les carrés 5 et 4, le carré de taille 4 n'est pas prolongeable.
– Avec les carrés 4, 3 et 2, c'est le carré de taille 2 qui n'est pas prolongeable.
– La seule façon de combler le plateau de largeur 9 est donc d'utiliser le carré de taille 9.

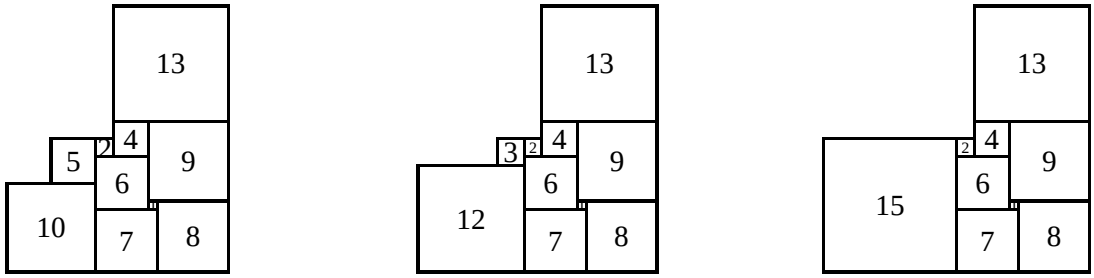


Nous pouvons continuer cette dernière configuration soit en prolongeant le carré de taille 6 verticalement soit en le prolongeant horizontalement.

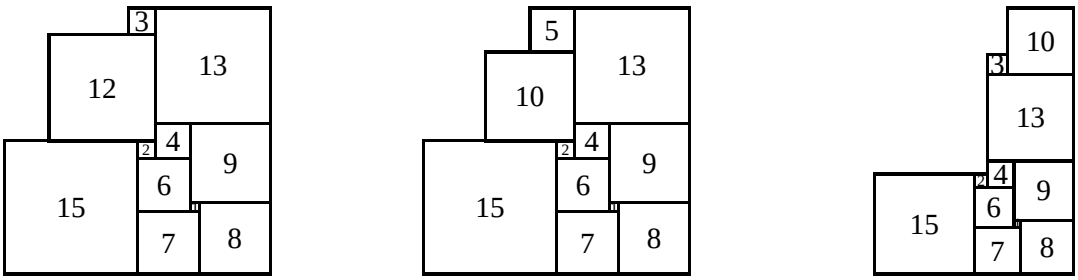
Sous-cas 1.2.1 - Prolongement vertical du carré 6 L'unique façon de réaliser ce prolongement est d'utiliser les carrés 2 et 4. Le plateau de largeur 13 ainsi généré ne peut être prolongé à gauche, il faut donc le prolonger verticalement avec les carrés 10 et 3 ou avec le carré de taille 13.



- Dans le premier cas, le carré de taille 10 n'est pas prolongeable.
- Il nous reste donc la deuxième configuration dans laquelle le carré de taille 2 ne peut être prolongé verticalement. Pour effectuer le prolongement horizontal on peut alors utiliser les carrés $\{5, 10\}$, $\{3, 12\}$ ou $\{15\}$.



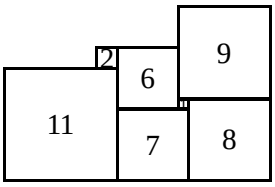
- Dans la première configuration, le carré de taille 5 est non prolongeable.
- Dans le deuxième cas, c'est le carré de taille 12 qui n'est pas prolongeable.
- Il faut donc utiliser un carré de taille 15 pour faire le prolongement. Regardons alors les prolongements possibles du carré de taille 13.

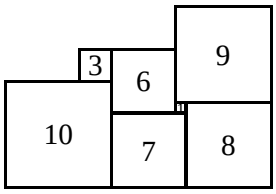


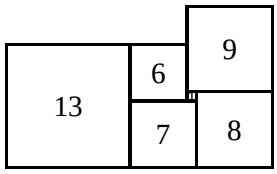
- Dans le premier cas de figure, le carré de taille 12 est non prolongeable.
- Dans le deuxième cas, c'est le carré de taille 10 qui est non prolongeable.
- Enfin, dans le dernier cas, c'est le carré de taille 10 qui n'est pas prolongeable.

Arrivant dans une impasse, nous savons que le carré de taille 6 ne peut pas se prolonger verticalement.

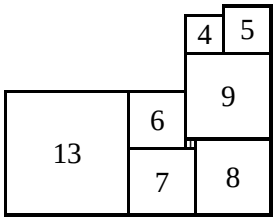
Sous-cas 1.2.2 - Prolongement horizontal du carré 6 Ce prolongement peut s'effectuer avec les carrés $\{11, 2\}$, $\{10, 3\}$, $\{13\}$.

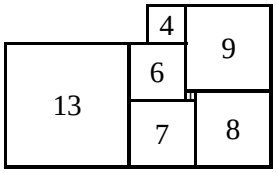




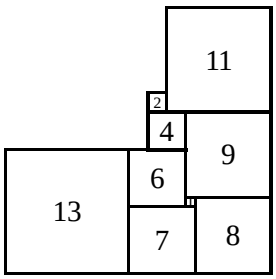


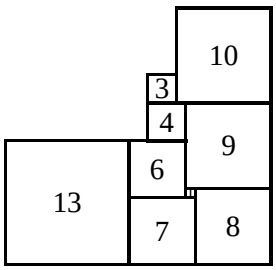
– En utilisant les carrés 11 et 2, le carré de taille 11 ne peut pas être prolongeable horizontalement et la seule façon de le prolonger verticalement est d'utiliser les carrés 5 et 4. Il reste alors un plateau de largeur 8 qui ne peut être prolongé.
– Dans cette deuxième configuration, le carré de taille 10 ne peut être prolongé horizontalement et ne peut être prolongé verticalement qu'en utilisant les carrés 5 et 2. Il reste alors un plateau de largeur 9 qui ne peut être prolongé.
– L'utilisation du carré de taille 13 reste donc la seule alternative. Regardons maintenant le prolongement du carré de taille 9.



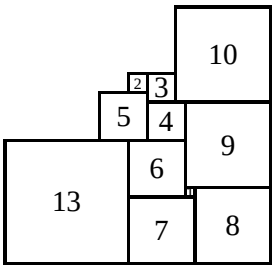


– Dans le premier cas, le carré de taille 5 est non prolongeable.
– Le carré de taille 4 placé dans la deuxième configuration ne peut être prolongé horizontalement. Pour le prolonger verticalement, on peut utiliser les carrés $\{2, 11\}$ ou les carrés $\{3, 10\}$.



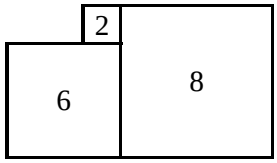


– Dans la première configuration, le carré de taille 2 n'est pas prolongeable.
– Dans le deuxième cas, le carré de taille 3 ne peut être prolongé verticalement et ne peut être prolongé qu'horizontalement avec les carrés 2 et 5.

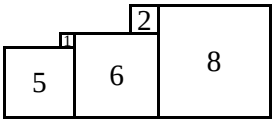


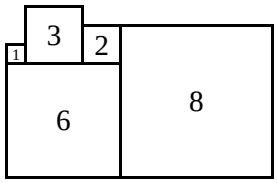
Enfin, dans cette dernière configuration, il est impossible de prolonger le plateau de largeur 5. Donc le prolongement du carré 8 ne peut pas se faire avec les carrés 7 et 1.

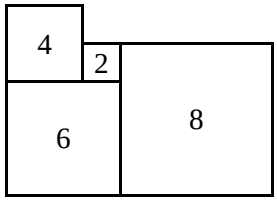
Cas 2 - Décomposition en $\{6, 2\}$



Nous allons nous intéresser ici au prolongement du carré de taille 6. Il existe trois façons d'effectuer ce prolongement qui sont les suivantes :

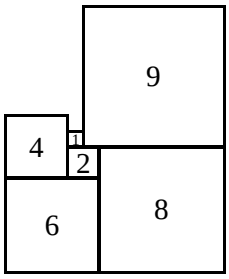


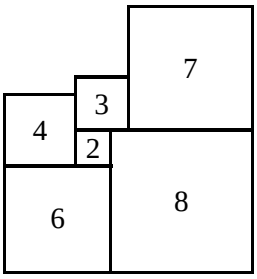


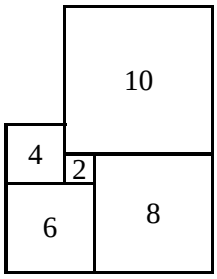


- La première configuration ne peut pas mener à une solution car le plateau de largeur 5 sur le carré de taille 1 n'est pas prolongeable.
- Dans le deuxième cas, le carré de taille 3 n'est pas prolongeable.
- Il faut donc placer le carré de taille 4 comme prolongement vertical du carré de taille 6.

Essayons maintenant de prolonger le plateau de largeur 10.

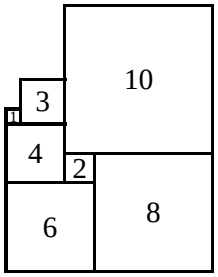


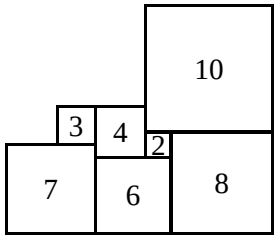


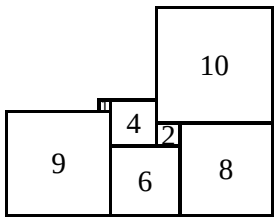


- L'utilisation des carrés 1 et 9 est impossible puisque le carré de taille 1 est non prolongeable.
- Dans la deuxième configuration, c'est le carré de taille 7 qui n'est pas prolongeable.
- La seule configuration possible est donc la troisième.

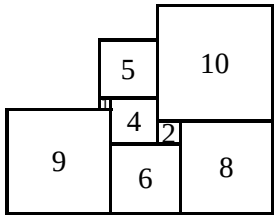
Regardons maintenant les différents prolongements possibles pour le carré de taille 4.





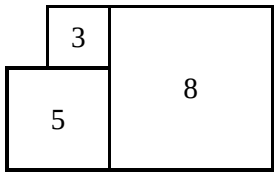


- Dans le premier cas, c'est le carré de taille 3 qui n'est pas prolongeable.
- Dans la deuxième configuration c'est le plateau de largeur 7 sur le plateau de taille 3 qui n'est pas prolongeable.
- Dans la troisième configuration, le seul prolongement possible pour le plateau de largeur 5 est de placer un carré de largeur 5.

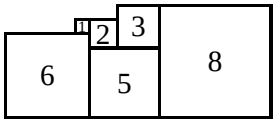


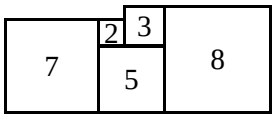
Dans cette configuration, il est impossible de prolonger le carré de taille 5 ce qui nous emmène dans une impasse et nous prouve que le carré de taille 8 ne peut être prolongé par les carrés 6 et 2.

Cas 3 - Décomposition en $\{5, 3\}$

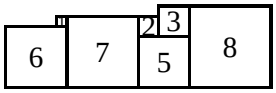


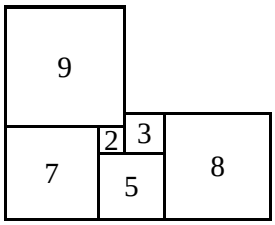
Dans cette nouvelle configuration, nous allons nous intéresser au prolongement du carré de taille 5. Celui-ci ne peut être prolongé horizontalement puisque nous avons vu qu'aucun carré de taille inférieure à 5 ne peut se trouver au bord d'une décomposition. Il faut donc effectuer un prolongement vertical avec le carré de taille 2. Ensuite, ce carré de taille 2 ne peut être prolongé verticalement, il faut donc le prolonger horizontalement. Deux choix s'offrent à nous :





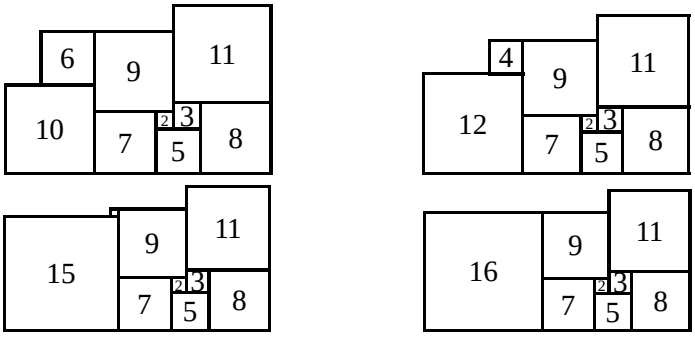
- Dans le premier cas, le plateau de largeur 3 sur le carré de taille 1 est non prolongeable.
- Dans la deuxième configuration, nous allons nous intéresser au prolongement du carré de taille 7.





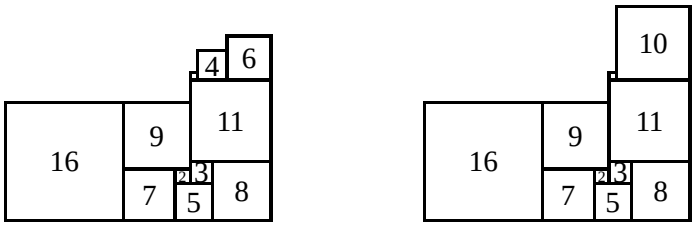
- Le premier cas pose problème car le carré de taille 6 n'est pas prolongeable.
- Dans le deuxième cas nous allons nous intéresser au plateau de largeur 11.

On ne peut pas prolonger verticalement le plateau de largeur 11 avec les carrés 1 et 10 car 1 ne pourra pas être prolongé à son tour. Pour les même raison, on ne peut pas utiliser les carrés 6, 4 et 1. Il faut donc utiliser le carré de taille 11. Maintenant, le carré de taille 9 n'étant pas prolongeable verticalement, intéressons nous à son prolongement horizontal. Pour cela, 4 configurations se présentent.

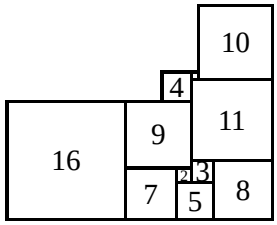


– Si on utilise les carrés 10 et 6, le carré de taille 11 n'est pas prolongeable.
– L'utilisation des carrés 12 et 4 nous empêche de prolonger le carré de taille 12.
– Le placement des carrés 15 et 1 nous empêche de prolonger le carré de taille 11.
– Le placement du carré de taille 16 est notre seule solution.

Regardons maintenant les différents prolongements du carré de taille 11.

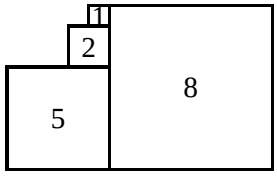


– Dans le premier cas, le carré de taille 6 n'est pas prolongeable.
– Dans le deuxième cas, le carré de taille 1 peut uniquement être prolongé par le carré de taille 4.



Arrivé à ce point, le carré de taille 4 ne peut pas être prolongé ce qui nous mène à une impasse. Donc le carré de taille 8 ne peut être prolongé par les carrés 5 et 3.

Cas 4 - Décomposition en $\{5, 2, 1\}$



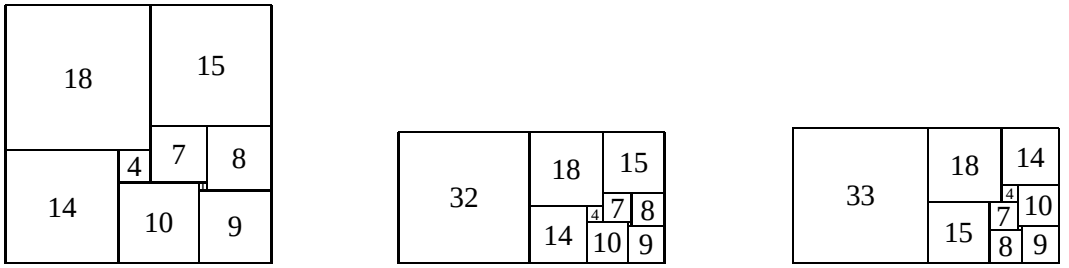
Ce dernier cas de prolongement pour le carré de taille 8 est impossible car le carré de taille 2 n'est pas prolongeable. Avec cette dernière configuration, nous avons terminé de montrer que le carré de taille 8 ne peut pas se trouver dans le coin d'une décomposition en carrés entiers.

3.7.5 Conclusion

Nous avons donc montré qu'il était impossible de réaliser une décomposition en carrés entiers ayant un carré de taille inférieure à 9 dans un des coins.

Propriété 8 *Dans une décomposition en carrés de tailles entières, aucun carré de taille 1 à 8 ne peut se trouver dans le coin.*

De plus, parmi les solutions que nous avons obtenues il existe des solutions avec un carré de taille 9 dans le coin de la décomposition en carrés entiers (figure 3.5). On remarquera que les deux derniers rectangles peuvent être trouvés à partir du premier en lui ajoutant un carré de taille 32 ou 33.



33×32, 9 Carrés 65×32, 10 Carrés 65×33, 10 Carrés

FIG. 3.5 – Décompositions connues ayant le plus petit carré dans le coin (de taille 9)

Si on ajoute ce résultat au résultat précédent, à savoir qu'il ne peut y avoir de carré inférieur à 5 sur le bord d'une décomposition entière, on obtient les gains suivants sur l'énumération du côté de la construction (tableau 3.4).

	Enumération Totale		*Tailles Bord > 4 et Coin > 8*		*Gain (Rapport)*	
Taille	*Configurations*	*Temps (ms)*	*Configurations*	*Temps (ms)*	*Configurations*	*Temps (ms)*
40	751 491	850	527	0	1 425	-
50	10 759 321	13 650	5 425	0	1 983	-
60	134 264 861	183 950	48 849	20	2 748	9 197
70	1 488 522 537	2 148 990	404 861	190	3 676	11 310
80	14 627 858 515	22 390 120	3 064 459	1 590	4 773	14 081
90			22 154 415	12 170		
100			148 550 255	88 260		
110			957 379 705	598 910		
120			5 836 497 609	3 816 990		

TAB. 3.4 – Gain obtenu sur l'énumération initiale en sachant que le plus petit carré au bord a une taille de 5 et que le plus petit carré dans le coin a une taille de 9.

Chapitre 4

Enumération exhaustive de décompositions entières

4.1 Idée générale

Cette approche du problème est en général la première qui vient à l'esprit. La méthode basée sur l'énumération de graphes nous donne, pour un nombre de carrés donnés, toutes les décompositions. Ici, nous allons plutôt nous fixer la taille du carré que nous souhaitons décomposer et énumérer toutes les décompositions possibles.

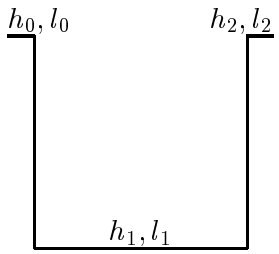


FIG. 4.1 – Etat initial de l'algorithme

Pour effectuer cette énumération, il suffit de partir de la configuration suivante (figure 4.1) avec $l_0 > 0$, $l_2 > 0$, $l_1 = taille\_carre$, $h_0 > taille\_carre$, $h_2 > taille\_carre$ et $h_1 = 0$ puis on effectue les opérations suivantes jusqu'à trouver une solution :

1. Trouver un plateau p_i entouré de plateaux plus élevés (tel que $h_{i-1} > h_i$ et $h_{i+1} > h_i$) ;
2. Décomposer ce plateau p_i avec des carrés distincts c_j^i tels que $\forall j, h_i + c_j^i < taille\_carre$.

4.2 Mise en œuvre

Ici, la méthode ne pose pas trop de problèmes pour son implantation. La structure de données que nous avons utilisée est une liste simplement chaînée, stockée dans un tableau. Ainsi, chaque élément du tableau est composé d'une structure avec 3 champs (largeur, hauteur, plateau suivant).

Une première remarque sur l'algorithme provient du fait que dans la première partie, lors de la recherche d'un plateau, il est très coûteux de trouver celui qui nécessitera le moins de décompositions possibles. Pour contourner ce problème il est possible de choisir celui qui est le

moins large. Ce choix n'est pas forcément le plus optimal mais permet une recherche vraiment plus rapide.

4.2.1 Première décomposition

La première décomposition (celle du plateau initial figure 4.1) est différente car elle permet d'effectuer de précieuses optimisations. En effet, la première observation permet de gagner un facteur 2 juste en évitant les symétries de cette décomposition. En effet, si la première décomposition est symétrique, nous retrouverons les mêmes solutions à une symétrie verticale près.

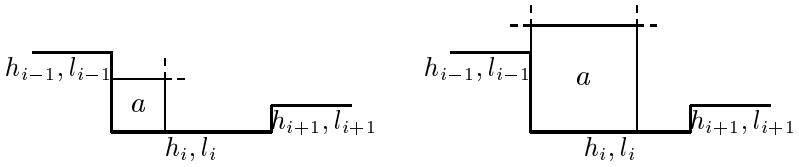
Pour cette première décomposition, il faut bien sûr profiter des propriétés 7 et 8 qui nous permettent d'éviter les carrés de tailles inférieures à 5 sur ce bord et inférieures à 9 dans les coins.

Nous cherchons ici à énumérer tous les carrés parfaits ayant une taille fixée. L'algorithme nous permettra aussi de trouver des décompositions parfaites de rectangles. L'algorithme interdisant de placer un carré au dessus du carré à décomposer, nous ne trouverons que des rectangles plus larges que hauts. Donc, la décomposition initiale fait partie du côté le plus large.

Une dernière optimisation permet de nous restreindre à traiter uniquement les décompositions initiales ayant au moins 3 carrés, nous ne perdrons aucune solution d'après la propriété 5. Nous évitons ainsi l'énumération de décompositions initiales composées de 2 carrés. Cette optimisation semble ne pas être significative mais lors des tests, nous avons obtenu un facteur de gain de l'ordre de 2,5.

4.2.2 Décompositions internes

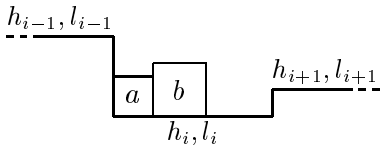
L'optimisation que nous allons aborder est proche de la propriété 2. Lors de la deuxième partie de l'algorithme (la décomposition d'un plateau), il est possible de limiter le nombre de décompositions que l'on peut générer. Voici un exemple de la forme que nous devons décomposer :



Nous allons chercher à décomposer le plateau p_i de la gauche vers la droite. Le nouveau carré a que nous cherchons à placer a plusieurs propriétés nécessaires pour son prolongement :

1. si $a < h_{i-1} - h_i$ alors le carré a doit être décomposable en somme d'entiers différents pour être prolongeable à droite ou verticalement.
2. si $a > h_{i-1} - h_i$ alors soit a est décomposable en somme d'entiers différents (prolongement vertical ou à droite), soit $a - (h_{i-1} - h_i)$ est une taille non utilisée ou alors décomposable (prolongement à gauche).

Cette optimisation n'est pas trop coûteuse et permet de couper l'arbre de recherche assez tôt. Le facteur de gain est ici de l'ordre de 2. Ce test est bien sûr aussi effectué sur les carrés suivants placés sur ce même plateau.



Une dernière optimisation peut être effectuée puisque lorsqu'on place de nouveaux carrés, on peut former des plateaux qui sont en contrebas, comme c'est le cas sur le schéma précédent. Puisque le plateau p_i était le moins large, alors forcément, le plateau de largeur a généré sur le schéma est encore plus petit. On connaît donc à cet instant les deux plateaux les plus petits puisqu'il est possible que le plateau en cours de décomposition ne soit pas terminé. Il faut donc comparer a avec $l_i - a - b$ et continuer avec le plateau le plus petit.

4.3 Algorithmes

Nous allons décrire à l'aide de pseudo-codes les différentes parties de l'algorithme. On décompose la méthode en trois parties qui constituent l'initialisation, la recherche d'un plateau à décomposer et enfin la décomposition de celui-ci. La procédure $min\_decomposable()$ renvoie

Algorithme 7 $initialisation(taille\_carre)$

$h_0 = h_2 = taille\_carre + 1$
$h_1 = 0$
$l_0 = l_2 = 1$
$l_1 = taille\_carre$
CALL $decompose(p_1)$

Algorithme 8 $plateau\_suivant()$

$l_{min} = taille\_carre + 1$
$i = 1$
while $\neg dernier\_plateau(p_i)$ **do**
  if $h_{i-1} > h_i \wedge h_i < h_{i+1} \wedge l_i < l_{min}$ **then**
    $p_{min} = p_i$
    $l_{min} = l_i$
  $i = i + 1$
if $l_{min} = taille\_carre$
  then if $h_{min} = taille\_carre$
    then <carré trouvé>
    else
      <rectangle trouvé>
      CALL $decompose(p_{min})$
  else CALL $decompose(p_{min})$

en réalité la somme des 2 carrés non encore placés les plus petits. Ainsi on sait qu'il n'existe aucune décomposition possible pour un carré ayant une taille plus petite que cette valeur.

La dernière partie de l'algorithme ($decompose(p_i)$) ne tient pas compte de toutes les améliorations définies précédemment car pour les mettre en œuvre, plutôt que de rajouter des tests dans la procédure, il est plus simple et plus efficace de créer 5 procédures différentes pour la décomposition qui nous permettent de savoir dans quel état on se trouve :

1. $decompose\_base\_debut()$ place le premier carré de la base. On teste ici que le premier carré est plus grand que 8. On appelle ensuite $decompose\_base\_suite()$ pour placer les carrés suivants.
2. $decompose\_base\_suite()$ place les carrés suivants de la base. C'est dans cette procédure que l'on vérifie que la base comporte plus de 2 carrés. Si on place le dernier carré de la base, il doit être plus grand que 8, sinon il doit être plus grand que 4. De plus le dernier

Algorithme 9 $decompose(p_i)$

if $taille\_non\_utilisee(l_i) \wedge h_i + l_i \leq taille\_carre$ **then**
  <sauver la configuration>
  <placer le carré de taille l_i>
  if $h_i = h_{i+1}$ **then** <inclure le plateau p_{i+1} au plateau p_i>
  if $h_i = h_{i-1}$ **then** <inclure le plateau p_i au plateau p_{i-1}>
  CALL $plateau\_suivant()$
  <restaurer la configuration>
if $taille\_non\_utilisee(h_{i-1} - h_i) \wedge h_{i-1} - h_i < l_i$ **then**
  <placer le carré de taille $h_{i-1} - h_i$>
  CALL $decompose(p_i)$
  <enlever le carré>
for $taille = min\_decomposable()$ **to** $MIN(l_i - 1, taille\_carre - h_i)$ **do**
  if $taille\_non\_utilisee(taille) \wedge taille \neq h_{i-1} - h_i$ **then**
    <placer le carré de taille $taille$ sur le plateau p_i>
    CALL $decompose(p_{i+1})$
    <enlever le carré>

carré placé doit être plus grand que le premier pour éviter les symétries. Si on place le dernier carré on appelle $plateau\_suivant()$ sinon on appelle $decompose\_base\_suite()$.

3. $decompose\_debut()$ place le premier carré d'une décomposition (hors la base). C'est la procédure appelée par $plateau\_suivant()$. Si on termine le plateau, alors on appelle $plateau\_suivant()$, sinon, si le plateau généré est plus bas que le plateau précédent ($taille + h_i < h_{i-1}$) on appelle $decompose\_suite\_bas()$ sinon on appelle $decompose\_suite\_haut()$.
4. $decompose\_suite\_haut()$ place les carrés suivants d'une décomposition (hors la base). On fait pratiquement comme dans $decompose\_debut()$ mais on sait ici que le plateau généré ne peut se trouver à la même hauteur que le précédent (sinon les carrés ont la même taille).
5. $decompose\_suite\_bas()$ place les carrés suivants d'une décomposition (hors la base). Ici c'est comme dans $decompose\_suite\_haut()$ mais on effectue un test supplémentaire. Puisque on est dans cette procédure plutôt que dans $decompose\_suite\_haut()$, c'est que $h_{i-2} > h_{i-1}$. Donc, si on place un carré de taille $taille$ et si $h_{i-1} < h_i + taille$ c'est que le plateau p_{i-1} est en contrebas, il peut donc être intéressant de le décomposer s'il est plus petit que le reste du plateau à décomposer. Donc, si $h_{i-1} < h_i + taille$ et $l_{i-1} < l_i - taille$ on appelle $decompose\_debut(p_{i-1})$ sinon on fait exactement comme $decompose\_suite\_haut()$.

4.4 Résultats numériques

Cette méthode a l'avantage de nous donner, pour une taille l donnée, toutes les décompositions de carrés de taille l. Durant cette recherche, il nous est aussi possible de trouver toutes les décompositions de rectangles dont le plus grand côté a pour longueur l. Cependant, le tableau 4.1 nous montre qu'il est très difficile d'atteindre des tailles importantes.

D'après les résultats obtenus, nous essayons un peu plus d'un million de carrés par seconde, ce qui revient à placer un carré en moins de 200 cycles du processeur (puisque nous travaillons sur un Pentium Pro 200) ce qui est assez rapide.

Ce qui peut être aussi intéressant, c'est de regarder la répartition du temps de résolution pour chaque procédure. Cette information nous est donnée dans le tableau 4.2.

Jasper Dale Skinner nous donne les résultats suivants :

Taille	*Nb de carrés*	*Nb de rectangles*	*Nb de carrés essayés*	*Temps*	*Taille*	*Nb de carrés*	*Nb de rectangles*	*Nb de carrés essayés*	*Temps*
33	0	1	3 325	10ms	106	0	7	16 205 316 088	3h 45m
57	0	1	1 221 580	1s 50ms	107	0	2	18 975 195 448	4h 23m
65	0	5	7 010 710	6s 70ms	108	0	0	31 395 647 078	7h 25m
66	0	1	8 452 428	7s 340ms	109	0	2	27 594 384 924	6h 32m
69	0	1	15 183 358	13s 170ms	110	3	1	33 889 697 521	8h 00m
75	0	1	54 033 532	47s 110ms	111	0	4	42 269 372 444	9h 53m
79	0	17	105 921 946	1m 31s	112	1	19	70 505 370 617	16h 46m
80	0	16	161 031 631	2m 21s	113	0	4	54 708 939 871	12h 34m
81	0	1	164 473 928	2m 22s	114	0	1	71 293 288 928	16h 31m
82	0	1	188 838 506	2m 43s	115	0	7	82 455 672 712	19h 25m
83	0	1	227 942 584	3m 17s	116	0	0	138 052 535 037	1j 09h
84	0	1	377 352 341	5m 33s	117	0	1	121 681 656 159	1j 05h
88	0	2	802 688 867	11m 52s	118	0	4	132 215 681 378	1j 06h
89	0	2	706 823 618	10m 12s	119	0	4	155 885 020 376	1j 11h
91	0	1	1 028 930 221	14m 52s	120	1	8	279 883 387 586	2j 19h
94	0	1	1 780 401 552	25m 42s	121	0	4	227 217 825 485	2j 04h
97	0	6	3 103 227 063	42m 50s	122	0	3	261 149 002 490	2j 11h
98	0	7	3 757 627 977	51m 56s	123	0	4	350 428 507 878	3j 10h
99	0	2	4 828 332 800	1h 07m	124	0	1	525 666 896 064	5j 08h
100	0	1	7 402 031 287	1h 45m	125	0	4	453 948 597 080	4j 08h
101	0	1	6 433 908 216	1h 28m	126	0	3	554 149 238 295	5j 08h
102	0	0	8 402 203 816	1h 56m	127	0	3	622 102 073 389	5j 22h
103	0	3	9 187 017 851	2h 06m	128	0	8	1 102 598 709 175	11j 03h
104	0	5	15 568 520 040	3h 48m	129	0	4	961 771 713 220	9j 08h
105	0	4	15 460 953 012	3h 47m	130	0	17	2 351 443 537 580	23j 22h

TAB. 4.1 – Solutions de l'énumération exhaustive

Nom de la procédure	*% du temps passé*
plateau_suivant()	25,76 %
decompose_debut()	23,51 %
decompose_suite_bas()	20,61 %
decompose_suite_haut()	20,44 %
decompose_base_suite()	9,69 %
decompose_base_debut()	0 %

TAB. 4.2 – Répartition du temps pour chaque procédure

– Les plus petites tailles pour les carrés parfaits non composés (et le plus petit ordre possible pour chaque) sont 110 (22), 112 (21), 120 (24), 139 (22), 140 (23).
– Les plus petites tailles pour les carrés parfaits composés (et le plus petit ordre possible

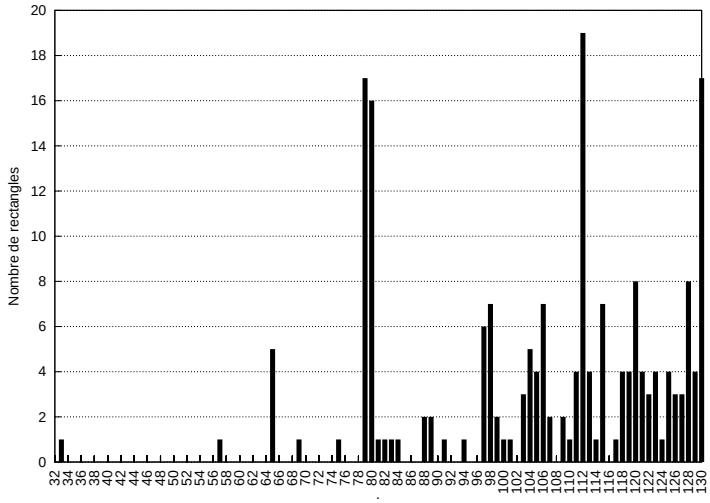


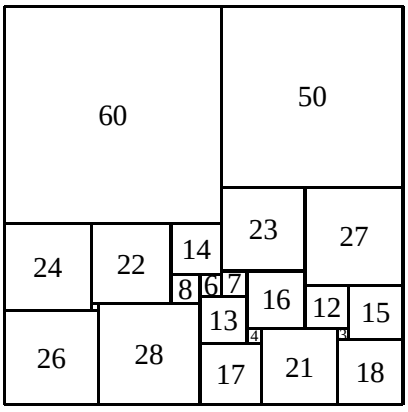
FIG. 4.2 – Répartition des découpages de rectangles en fonction de la taille énumérée

pour chaque) sont 175 (24), 235 (25), 288 (26), 324 (27), 325 (27).
Ces résultats semblent plus complets que ceux que nous avons trouvé mais la grande différence provient du fait qu'ils sont issus de la méthode par l'énumération de graphe en prenant les décompositions de plus petite taille. De ce fait, cette façon de faire ne nous garantit pas que les solutions sont bien minimales en taille.

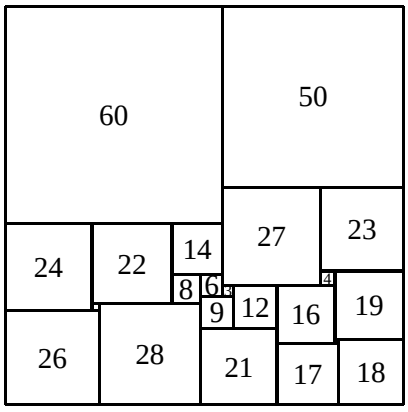
Notre méthode par énumération exhaustive quant à elle a le mérite de garantir la minimalité des solutions trouvées en terme de taille.

4.5 Quelques solutions

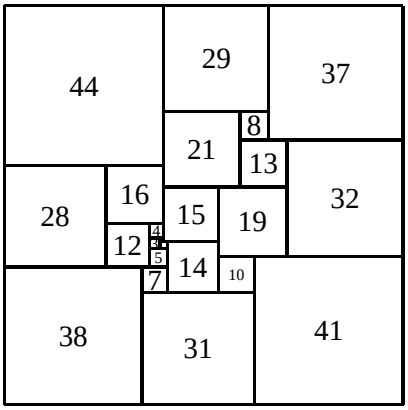
Nous donnons ici l'ensemble de tous les carrés parfaits trouvés.



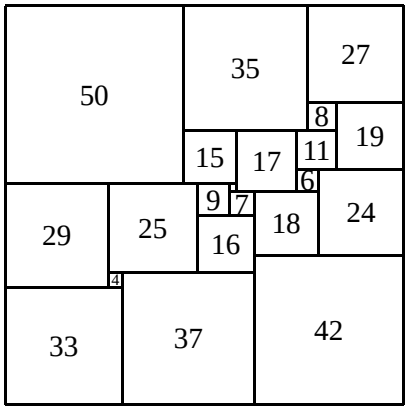
110 × 110, 22 Carrés



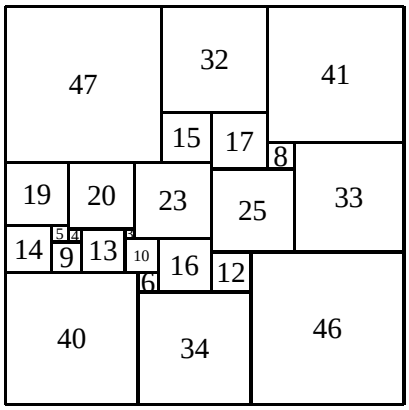
110 × 110, 22 Carrés



110×110, 23 Carrés



112×112, 21 Carrés



120×120, 24 Carrés

Chapitre 5

Enumération sélective de décompositions entières

5.1 Idée générale

La méthode de construction que nous allons décrire dans cette partie vient d'une remarque sur la structure de la solution d'ordre 21 (figure 1.1) et a donné lieu à l'écriture d'un article accepté pour publication dans la revue «Discrete Applied Mathematics» [29].

Supposons connu le bas de notre décomposition, soit dans notre cas les carrés de tailles 33, 37 et 42 et essayons de voir s'il existe une méthode simple nous permettant de retrouver la solution d'ordre 21.

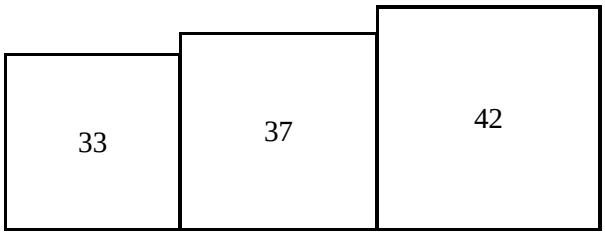


FIG. 5.1 – Configuration initiale

A partir de cette configuration (figure 5.1), il est possible de trouver le carré de taille 4 par simple différence entre les carrés de tailles 37 et 33. De la même manière, par différence entre le carré de taille 33 et celui de taille 4, on trouve le carré de taille 29. En réalité, plutôt que de parler de différences entre tailles de carrés, il est plus facile de parler de prolongements des bords des carrés. Par exemple, pour le carré de taille 4, il suffit de prolonger le dessus du carré de taille 37. Pour le carré de taille 29, on prolonge le côté gauche du carré de taille 33. A partir de ces remarques, définissons l'ensemble des règles qui nous permettent de trouver la solutions d'ordre 21.

L'idée que nous sommes en train de mettre en œuvre construit la décomposition du bas vers le haut en empilant des carrés les uns sur les autres. A chaque étape de cet algorithme, nous avons uniquement besoin de connaître le «dessus» de la construction. Notons h_i et l_i la hauteur et la largeur du plateau horizontal p_i. Définissons maintenant les deux règles qui vont permettre la construction de la solution d'ordre 21.

– Le prolongement horizontal (figure 5.2) qui consiste à ajouter un carré de taille $h_{i+1} - h_i$ si $h_{i+1} > h_i$ et de taille $h_i - h_{i+1}$ si $h_i > h_{i+1}$.

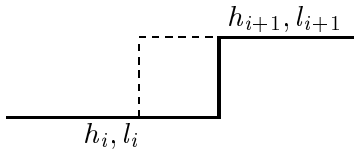


FIG. 5.2 – Prolongement horizontal

– Le prolongement vertical (figure 5.3) qui consiste à placer un carré sur la totalité du plateau i de longueur l_i.

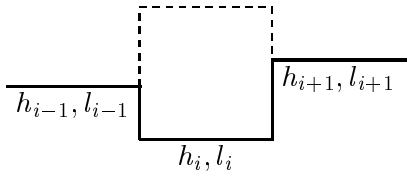


FIG. 5.3 – Prolongement Vertical

En utilisant ces deux règles, il est possible à partir de la configuration de la figure 5.1 d'obtenir la solution d'ordre 21 (figure 1.1). Voici dans quel ordre l'algorithme trouve les différents carrés : 33, 37, 42, 4, 29, 25, 50, 16, 9, 7, 2, 15, 18, 17, 6, 24, 11, 35, 8, 19, 27.

Le grand avantage de cette méthode réside dans le faible nombre de choix à chaque étape de l'algorithme. Ainsi, à partir d'une configuration initiale et en utilisant les règles définies précédemment, la construction se fait très rapidement.

5.2 Mise en œuvre

La méthode décrite semble simple mais pose quelques problèmes de mise en œuvre. Nous allons donc étudier chacun de ces problèmes en essayant d'y apporter la meilleure solution possible.

5.2.1 L'initialisation

Tout d'abord, le principal inconvénient de cette la méthode est dû à son initialisation. En effet, pour pouvoir appliquer les règles de construction, il faut avoir fixé le nombre de carrés sur un côté et aussi connaître leurs tailles. Le problème ici est important car l'énumération des configurations possibles est très coûteux. Le tableau 5.1 donne un aperçu du problème de l'initialisation de notre algorithme. On peut voir par exemple que pour un carré de taille 80×80, on a 14 627 858 515 configurations possibles pour l'initialisation de la méthode et le temps de génération de ces configurations est de 22 390 120 millisecondes, soit un peu plus de 6 heures !

Il faut savoir que nous cherchons à atteindre des tailles d'au moins 112 pour pouvoir trouver la solution des 21 carrés (figure 1.1). Si nous appliquons une méthode d'énumération totale comme celle-ci, nous aurons du mal à atteindre des tailles acceptables de carrés. Heureusement, il est possible de réduire grandement la complexité de l'énumération. En effet, si on s'intéresse aux décompositions utilisant uniquement des carrés ayant pour taille des entiers, l'étude des tailles minimales de carrés qui peuvent se trouver sur le bord et dans le coin d'une décomposition nous permet de diminuer grandement le coût de notre initialisation. Une autre information nous permet encore de pratiquement diviser par deux nos temps d'énumération

Taille	*Initialisations possibles*	*Temps (ms)*
6	11	0
40	751 491	850
50	10 759 321	13 650
60	134 264 861	183 950
70	1 488 522 537	2 148 990
80	14 627 858 515	22 390 120

TAB. 5.1 – Nombre de décomposition d'un entier en somme d'entiers différents

si on tient compte de la symétrie horizontale du problème. En modifiant la procédure d'énumération pour tenir compte de ces nouveaux résultats, on obtient les temps du tableau 5.2 qui sont bien plus raisonnables.

Taille	*Décompositions*	*Temps (ms)*
6	1	0
40	264	0
50	2 713	0
60	24 425	10
70	202 431	100
80	1 532 230	800
90	11 077 208	6 090
100	74 275 128	44 510
110	478 689 853	292 820
120	2 918 248 805	1 843 570

TAB. 5.2 – Nombre de décompositions d'entiers avec une taille minimale de 5 et une taille minimale de 9 dans le coin sans symétrie.

5.2.2 Amélioration de l'initialisation

Après avoir obtenu une configuration initiale, il est possible de la modifier légèrement pour obtenir une configuration plus intéressante. En effet, si on place deux plateaux supplémentaires sur les côtés de notre configuration, on obtiendra la figure 5.4.

Ces deux plateaux supplémentaire ont une largeur quelconque (non nulle) et une hauteur égale à la largeur de la base. Cette configuration est avantageuse car elle favorisera la construction de carrés par prolongement horizontal de ces deux nouveaux plateaux. Cette modification permet de trouver légèrement plus de solutions sans ralentir et sans complexifier la méthode de résolution.

5.2.3 Nombre de carrés à l'initialisation

Un autre paramètre très important de l'algorithme que nous pouvons introduire est le nombre de carrés utilisés lors de l'énumération des configuration de départ. En effet, il est intéressant de regarder la répartition des configurations initiales en fonction du nombre de carrés utilisés (tableau 5.3).

On peut voir par exemple que pour la décomposition d'un carré de taille 120×120, la majorité des initialisations possibles comportent 10 carrés. Maintenant, sachant qu'un carré a 4 côtés, il est intéressant de regarder dans les décompositions connues quel est le nombre de

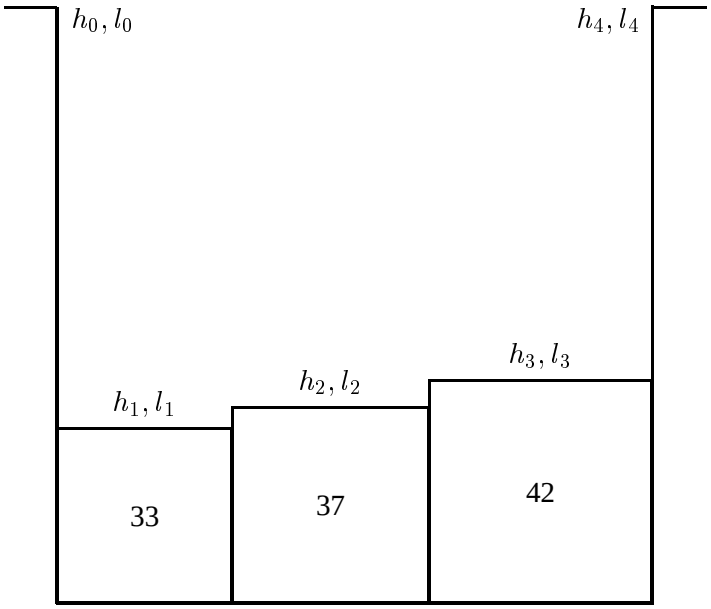


FIG. 5.4 – Etape 0 de l'algorithme

Nb de carrés utilisés	*Taille de la décomposition*									
	6	40	50	60	70	80	90	100	110	120
1	1	1	1	1	1	1	1	1	1	1
2		11	16	21	26	31	36	41	46	51
3		74	184	345	554	814	1 125	1 484	1 894	2 355
4		142	844	2 590	5 890	11 236	19 138	30 082	44 584	63 130
5		36	1 308	8 724	32 040	86 028	190 404	369 960	654 408	1 078 524
6			360	10 584	81 600	352 320	1 099 464	2 786 160	6 114 600	12 073 824
7				2 160	73 680	672 120	3 407 280	12 294 360	35 503 200	87 604 680
8					8 640	409 680	4 671 360	28 346 400	119 082 960	392 702 400
9							1 688 400	26 213 040	197 416 800	982 638 720
10								4 233 600	112 250 880	1 111 864 320
11									7 620 480	330 220 800

TAB. 5.3 – Répartition des configuration initiale en fonction du nombre de carrés utilisés.

carrés sur un côté. Si on regarde la majorité des décompositions existantes, elles ont très souvent un côté utilisant assez peu de carrés (entre 2 et 4). Si on se restreint aux décompositions ayant par exemple un côté avec 3 carrés, alors le nombre de configurations initiales sera grandement réduit. En réalité il faut regarder le rapport entre le nombre de configurations initiales et le nombre de configurations trouvées. Ce que l'on sait pour le moment c'est que le nombre de configurations initiales augmente beaucoup en fonction du nombre de carrés utilisés. Par exemple pour un carré de taille 120×120, on passe de 2 355 configurations pour 3 carrés à 1 111 864 320 configurations pour 11 carrés. Nous étudierons par la suite l'effet de ce paramètre sur l'efficacité de notre méthode.

5.2.4 Utilisation des règles de construction

La méthode que nous venons de décrire semble simple mais la plus grande difficulté est de savoir comment appliquer nos deux règles de construction de manière optimale. Cette partie est assez délicate car il faut éviter de retomber sur un état déjà obtenu précédemment pour ne pas dupliquer les résultats et augmenter le temps d'exécution de l'algorithme. D'un autre côté, il ne faut pas oublier de traiter certaines configurations.

Pour cela, nous allons utiliser les règles de construction de la gauche vers la droite. Par exemple, sur la figure 5.4, pour placer un nouveau carré, on va d'abord essayer un prolongement horizontal entre les plateaux p_0 et p_1, puis on essayera de prolonger verticalement le plateau p_1, puis un prolongement horizontal entre p_1 et p_2, puis le prolongement vertical de p_2, etc...

A chaque fois que l'on place un nouveau carré, si on essaye de placer un autre carré en partant du plateau le plus à gauche on risque de retomber sur une configuration déjà traitée. Par exemple, si on place comme premier carré, le carré de taille 4 par prolongement horizontal des carrés 33 et 37, par la suite il nous sera possible de placer le carré de taille 5 par prolongement horizontal des carrés 37 et 42. Si maintenant on commence par placer le carré de taille 5, puis celui de taille 4, on va retomber sur la même configuration. Si on ne fait pas suffisamment attention à ce problème, le nombre de configurations traitées plusieurs fois va être énorme et donc générer les même solutions, mais surtout le temps de résolution sera beaucoup plus important.

Une première méthode pour éviter la duplication des configurations est de mémoriser les états déjà traités. Cette méthode nécessite une place mémoire non négligeable et ralentit énormément la résolution.

La meilleure solution ici est de trouver une méthode d'utilisation des deux règles ne pouvant pas générer plusieurs fois les mêmes configurations. A chaque nouveau carré placé sur un plateau i, on ne continue pas la recherche en repartant du plateau le plus à gauche. On pourrai penser qu'il suffit de placer les nouveaux carrés en partant de l'emplacement du dernier carré placé, mais en plaçant un nouveau carré, on modifie la structure de la construction et on rajoute donc des prolongement possibles à gauche du dernier carré placé. Donc, après chaque placement, nous essayerons de placer un nouveau carré le plus à gauche possible qui n'était pas possible de trouver à partir de la configuration précédente.

Donnons un exemple à partir de la figure 5.4. Si on place le carré de taille 5 par prolongement horizontal entre les carrés 37 et 42, on modifie la largeur du plateau p_2 et p_3, donc le placement de ce nouveau carré va nous permettre de trouver un nouveau carré si on prolonge le plateau p_2 qui est donc maintenant de largeur $l_2 = 37 - 5 = 32$. Par contre le carré trouvé par prolongement horizontal entre les plateaux p_1 et p_2 n'est pas nouveau puisqu'il était possible de le trouver avant d'avoir placé ce nouveau carré. Donc dans ce cas précis, nous commençons la recherche pour l'emplacement d'un nouveau carré à partir du prolongement vertical du plateau p_2, puis nous essayons un prolongement horizontal entre p_2 et p_3 puis un prolongement vertical de p_3 et ainsi de suite.

Les différents cas pouvant se présenter sont inclus dans les algorithmes correspondants au prolongement horizontal et à celui d'un prolongement vertical.

5.2.5 Optimisations

Il est important de remarquer que cet algorithme ne permet pas d'augmenter le nombre de plateaux durant la construction. Ceci est vrai car nos deux règles de construction (le prolongement horizontal et vertical) ne créent pas de nouveaux plateaux, elles modifient juste la hauteur (prolongement vertical) ou la largeur (prolongement horizontal) de plateaux existants. Le nombre de plateaux ne fait donc que décroître jusqu'à atteindre un seul plateau, ce qui indique

que le programme a trouvé une nouvelle décomposition de carré. Cependant, cet algorithme permet également de trouver des découpages de rectangles.

Les deux règles de construction permettent de créer uniquement de nouveaux carrés qui sont la somme ou la différence de carrés déjà existants. Ainsi, si les carrés utilisés à l'initialisation de l'algorithme ont un diviseur commun, tous les nouveaux carrés seront aussi divisibles par celui-ci. Nous sommes en train de traiter une configuration déjà analysée auparavant à un multiple près. Pour y pallier, il suffit de calculer le plus grand commun diviseur des carrés initiaux. Si celui-ci est différent de 1, on ne fait pas la recherche sur cette configuration.

Ce test ne permet pas d'éliminer un nombre suffisant de cas pour faire gagner beaucoup de temps de calcul. Par contre, il permet de faire apparaître uniquement les solutions entières non divisibles.

Il existe aussi d'autres configurations initiales qui ne peuvent pas donner de résultats. Par exemple, celle de la figure 5.5 ne peux pas donner de solution car l'algorithme ne pourra jamais placer des carrés au dessus du carré de taille 15.

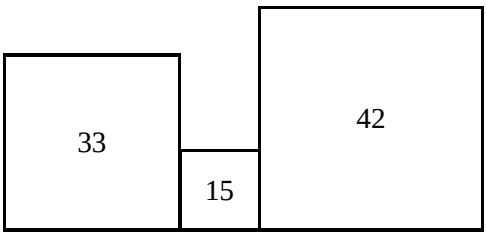


FIG. 5.5 – Exemple de configuration initiale sans solution

Pour généraliser, quel que soit le plateau i, si nous prenons (h_{i-1}, l_{i-1}), (h_i, l_i), (h_{i+1}, l_{i+1}) trois plateaux consécutifs issus de l'initialisation, alors la configuration est sans solution si on a $l_i < h_{i-1} - h_i$ et $l_i < h_{i+1} - h_i$. Cette remarque permet d'accélérer l'algorithme.

5.3 Algorithmes

Nous donnons ici les pseudo-codes basés sur l'algorithme décrit précédemment. Il y a deux grandes parties. La première (algorithme 10) est très simple et son but est de générer toutes les configurations initiales comme celle de la figure 5.4. Ensuite, il suffit de lancer la procédure de résolution. Cette étape initiale nécessite deux paramètres qui sont la taille du carré à décomposer ($taille\_carre$) et le nombre de carrés utilisés pour fabriquer la configuration initiale (nb_carres). L'algorithme de résolution ne nécessitant que la connaissance du

Algorithme 10 $resolution(taille\_carre, nb\_carres)$

$h_0 = h_{nb\_carres+1} = taille\_carre$
$l_0 = l_{nb\_carres+1} = 1$
<initialisation de l'énumération>
for each <énumération> **do**
  CALL $prolongement\_horizontal(p_0, p_1)$

dessus de la construction, cette première étape nous fournit un état du système représenté par deux vecteurs appelés h_i et l_i qui représentent la hauteur et la largeur de chaque plateau p_i ($i \in \{0 \dots nb\_carres + 1\}$). Maintenant en suivant les explications précédentes de l'algorithme et en faisant attention à quelques cas particuliers, il est possible d'écrire les deux procédures suivantes (le prolongement horizontal et le prolongement vertical). Ces deux procédures forment

Algorithme 11 $prolongement\_horizontal(p_i, p_{i+1})$

if $h_{i+1} > h_i$ **then**
  if $h_{i+1} - h_i < l_i \wedge taille\_non\_utilisee(h_{i+1} - h_i)$ **then**
    <placer le carré de taille $h_{i+1} - h_i$>
    if $h_{i-1} - h_i = l_i$ **then**
      if $\neg last\_plate(p_{i+1})$
        then CALL $prolongement\_vertical(p_{i+1})$
        else CALL $prolongement\_vertical(p_i)$
    <enlever le carré>
else if $h_i - h_{i+1} < l_{i+1} \wedge taille\_non\_utilisee(h_i - h_{i+1})$ **then**
  <placer le carré de taille $h_i - h_{i+1}$>
  if $\neg premier\_plateau(p_i)$
    then if $h_{i-1} \geq l_i + h_{i+1}$
      then CALL $prolongement\_horizontal(p_{i-1}, p_i)$
      else CALL $prolongement\_vertical(p_i)$
    else CALL $prolongement\_vertical(p_{i+1})$
  <enlever le carré>
if $\neg dernier\_plateau(p_{i+1})$ **then** CALL $prolongement\_vertical(p_{i+1})$

Algorithme 12 $prolongement\_vertical(p_i)$

if $taille\_non\_utilisee(l_i) \wedge h_i + l_i \leq taille\_carre$ **then**
  <sauver la configuration>
  $h' = h_{i-1}$
  $l' = l_{i-1}$
  <placer le carré de taille l_i>
  if $h_i = h_{i+1}$ **then** <ajouter le plateau p_{i+1} à p_i>
  if $h_{i-1} = h_i$ **then** <ajouter le plateau p_{i-1} à p_i>
  <tester si on a trouvé une solution>
  if $premier\_plateau(p_i)$
    then if $\neg dernier\_plateau(p_i)$ **then** CALL $prolongement\_horizontal(p_i, p_{i+1})$
    else if $h_i - l_i - h_{i-1} = l_{i-1}$
      then CALL $prolongement\_vertical(p_{i-1})$
      else if $h' = h_i \wedge h_{i-1} - h_i \leq l'$
        then if $\neg dernier\_plateau(p_i)$ **then** CALL $prolongement\_vertical(p_i)$
        else CALL $prolongement\_horizontal(p_{i-1}, p_i)$
  <restaurer la configuration>
if $h_{i-1} - h_i \leq l_i \vee h_{i+1} - h_i \leq l_i$ **then** CALL $prolongement\_horizontal(p_i, p_{i+1})$

le cœur de l'algorithme et tout le temps de calcul est passé ici. Elles correspondent aux deux règles principales énoncées précédemment : le prolongement horizontal et le prolongement vertical. Dans ces pseudo-codes, nous avons utilisés un tableau pour représenter la configuration des plateaux à un instant donné mais dans une réelle implémentation, il est plus intéressant d'utiliser une liste chaînée stockée dans un tableau puisque la suppression et l'insertion d'un plateau est beaucoup plus rapide.

5.4 Résultats numériques

Tout d'abord, regardons quelles sont les procédures qui sont très coûteuses en temps :

Nom de la procédure	*% du temps passé*	*% d'appels*
prolongement_vertical()	91 %	59 %
resolution()	8 %	0 %
prolongement_horizontal()	1 %	41 %

TAB. 5.4 – Répartition du temps pour chaque procédure

Nous avons cherché à obtenir de très nombreux résultats, c'est pour cela que nos temps de calcul sont relativement longs. Nous avons aussi tenu à séparer les résultats en fonction du nombre de carrés utilisés à la base. Ceci nous permettra ensuite d'analyser l'effet de ce paramètre dans les résultats obtenus. Avant de faire cette distinction, voici tout d'abord les résultats obtenus lorsque le nombre de carrés dans la configuration initiale n'est pas fixé.

5.4.1 Base quelconque

Les résultats obtenus avec un nombre quelconque de plateaux et en énumérant la taille des carrés à découper entre 6 et 125 sont les suivants :

- 4 carrés simples et aucun carré composé (4 doublons retirés) ;
- 49 rectangles simples et 14 rectangles composés (14 doublons retirés) ;
- le premier carré parfait trouvé est un carré de taille 110×110 d'ordre 22 après 66 heures ;
- 43 154 886 289 configurations initiales traitées sur 43 161 537 427 générées, sur lesquelles 3 470 155 135 356 carrés ont été placés ;
- le calcul a pris 1 119 heures (soit environ 46 jours) sur un Pentium Pro 200 sous Linux.

Comme nous l'avions prévu, les résultats sont pratiquement inexistants. Les configurations initiales générées sont trop nombreuses, contiennent beaucoup de carrés et ne donnent pratiquement pas de solutions. Si on observe les solutions, seules les configurations initiales avec un nombre assez petit de carrés donnent des résultats.

Si on utilise le même algorithme mais en énumérant uniquement les configurations initiales ayant un nombre de carrés fixé assez petit on obtiendra certainement de meilleurs résultats.

5.4.2 Base avec 4 carrés

Les résultats obtenus avec 4 plateaux en énumérant la taille des carrés à découper entre 6 et 450 sont les suivants :

- 262 carrés simples et 2 carrés composés (19 doublons retirés) ;
- 9 772 rectangles simples et 1 404 rectangles composés (727 doublons retirés) ;
- le premier carré parfait trouvé est un carré de taille 110×110 d'ordre 22 après 58 secondes ;
- 609 822 310 configurations initiales traitées sur 657 876 250 générées, sur lesquelles 2 775 894 996 818 carrés ont été placés ;

– le calcul a pris 580 heures (soit environ 24 jours) sur un Pentium Pro 200 sous Linux.

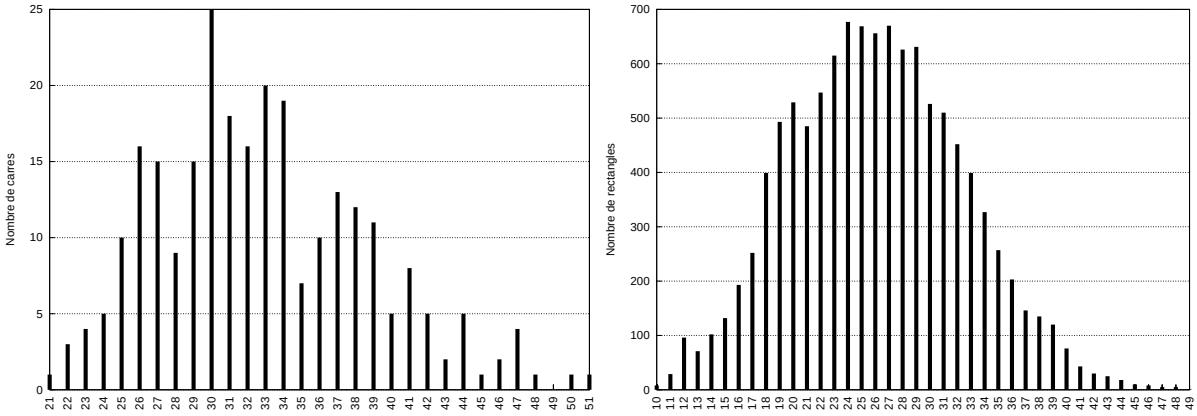


FIG. 5.6 – Répartition des solutions en fonction du nombre de carrés utilisés (avec 4 plateaux).

Ici les résultats commencent à devenir intéressants mais demandent encore beaucoup de temps de calcul puisque l'on trouve environ 1 carré toute les 2 heures. De plus, nous avons surtout trouvé des décompositions de rectangles en nombre important plutôt que des décompositions de carrés.

5.4.3 Base avec 3 carrés

Les résultats obtenus avec 3 plateaux en énumérant la taille des carrés à découper entre 6 et 1 500 sont les suivants :

– 7 999 carrés simples et 6 carrés composés (28 doublons retirés) ;
– 8 842 rectangles simples et 1 181 rectangles composés (1 166 doublons retirés) ;
– le premier carré parfait trouvé est celui d'ordre 21 (cf. figure 1.1) après 2,2 secondes ;
– 268 785 136 configurations initiales traitées sur 223 784 787 générées, sur lesquelles 2 020 806 935 885 carrés ont été placés ;
– le calcul a pris 402 heures (soit environ 17 jours) sur un Pentium Pro 200 sous Linux.

Ces résultats sont certainement les plus intéressants car ils contiennent un nombre important de carrés et de rectangles. Vu le nombre important de solutions, nous avons tracé différentes courbes pour montrer le comportement de notre algorithme. Dans les graphiques de la figure 5.7, on peut voir le nombre de carrés trouvés en fonction de leur ordre. On peut remarquer

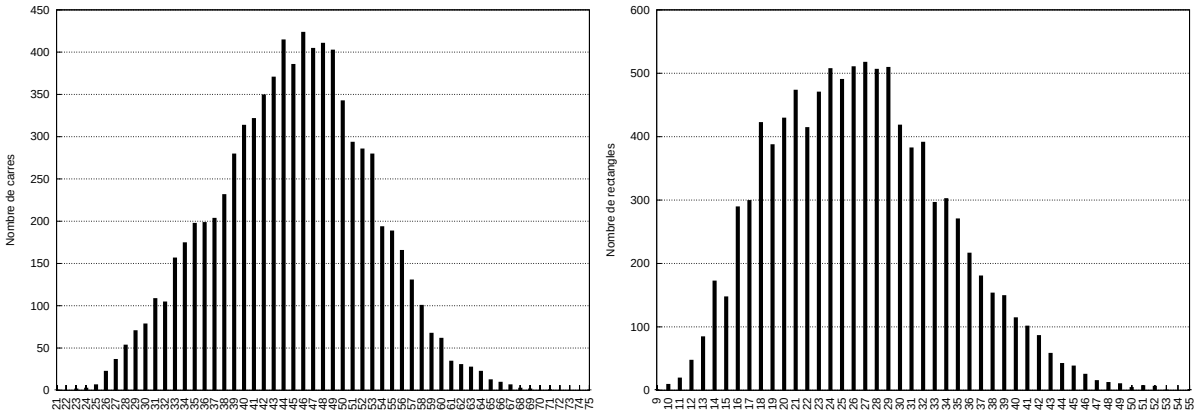


FIG. 5.7 – Répartition des solutions en fonction du nombre de carrés utilisés (avec 3 plateaux).

une courbe ayant la forme d'une «cloche» qui s'étend vers les ordres supérieurs lorsque l'algorithme continue. Plus la taille à décomposer est importante, plus les solutions trouvées ont un nombre important de carrés dans leurs décompositions. La figure 5.8 représente l'efficacité de

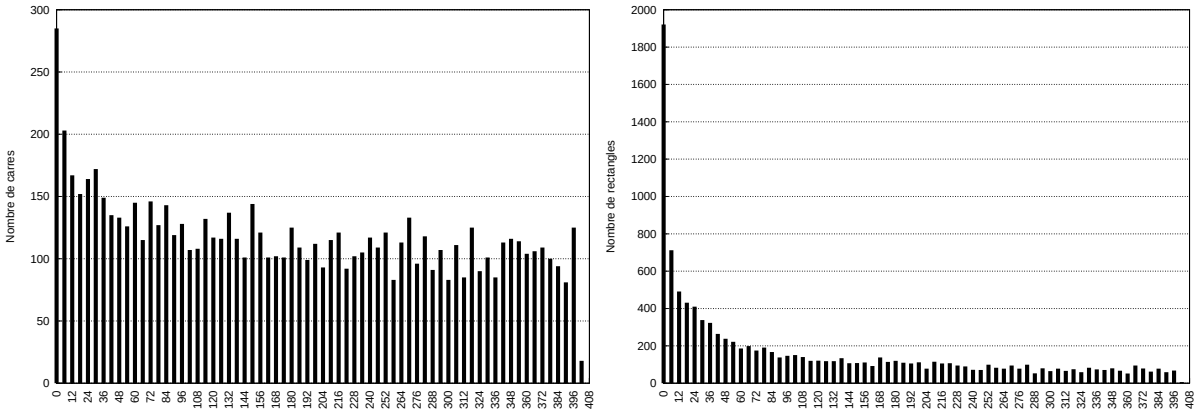


FIG. 5.8 – Répartition des solutions dans le temps (en heures de calcul, avec 3 plateaux).

l'algorithme dans le temps. On voit ici clairement que le nombre de solutions trouvées décroît légèrement dans le temps. On obtient ici une moyenne de 20 carrés parfaits et 25 rectangles par heure. Dans la figure 5.9 on remarque que le nombre de solutions augmente lorsque la taille du

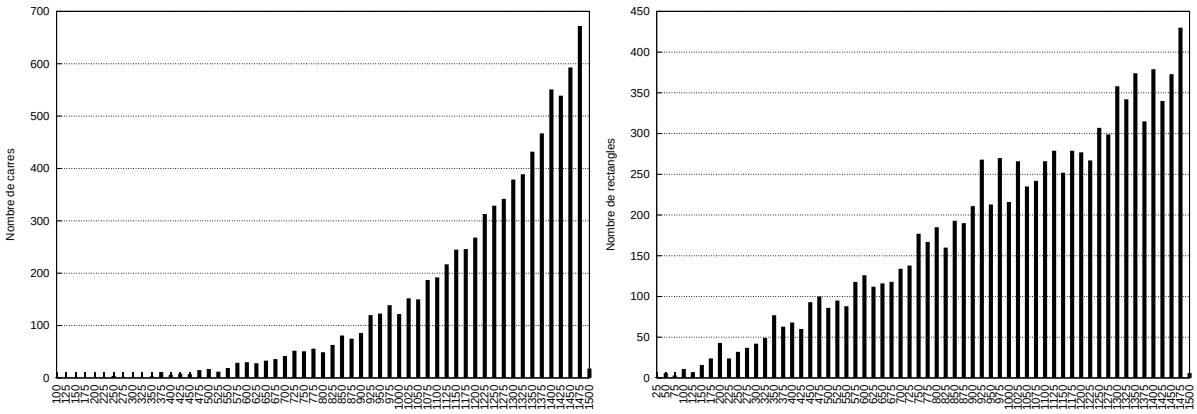


FIG. 5.9 – Répartition des solutions en fonction de la taille de la décomposition (avec 3 plateaux).

carré à décomposer augmente. Ceci est tout a fait normal mais ce qui est surprenant c'est que le nombre de carrés trouvés augmente plus rapidement que celui des rectangles. C'est certainement pour cette raison que précédemment nous avons surtout trouvé des rectangles plutôt que des carrés. Enfin, la figure 5.10 nous donne pour les rectangles trouvés la répartition des solutions en fonction du rapport des côtés. On peut remarquer ici que nous ne trouvons aucun rectangle ayant un rapport des côtés inférieur à 0,43.

5.4.4 Base avec 2 carrés

Les résultats obtenus avec 2 plateaux et en énumérant la taille des carrés à découper entre 6 et 300.000 sont les suivants :

- 23 610 carrés simples et 4 carrés composés (aucun doublon) ;
- Aucun rectangle ;

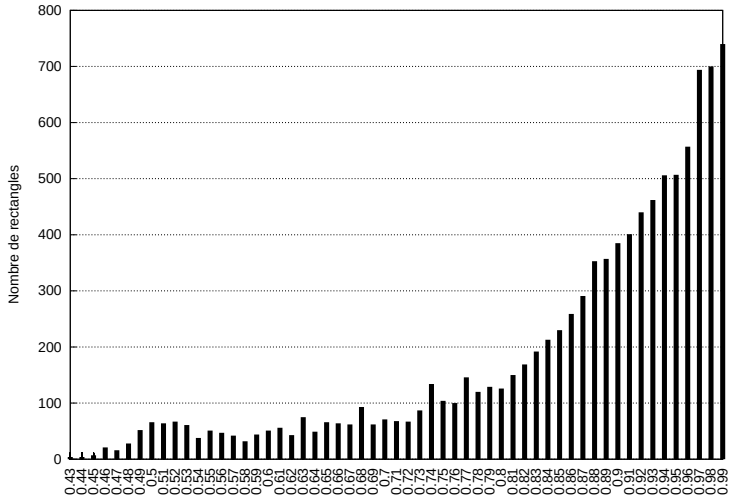


FIG. 5.10 – Nombre de rectangles trouvés en fonction du rapport des côtés (avec 3 plateaux).

- le premier carré parfait trouvé est un carré de taille 2696×2696 d'ordre 39 après 11,5 secondes ;
- 13 676 827 142 configurations initiales traitées sur 22 497 450 072 générées, sur lesquelles 217 444 649 052 carrés ont été placés ;
- le calcul a pris 48 heures sur un Pentium Pro 200 sous Linux.

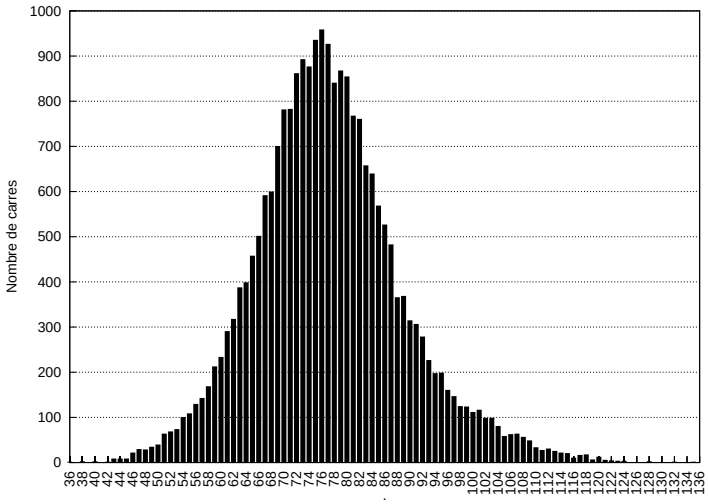


FIG. 5.11 – Répartition des solutions en fonction du nombre de carrés utilisés (avec 2 plateaux)

Ici, on a une moyenne de 490 carrés parfaits par heure!! Le seul inconvénient ici est que les carrés trouvés sont très vite de grande taille et d'ordre important.

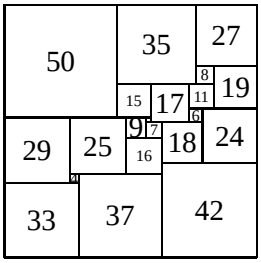
5.4.5 Conclusion

Cet algorithme n'est pas complet car nous ne trouvons qu'une partie des résultats pour une taille donnée mais en revanche, nous trouvons une énorme quantité de solutions ayant des ordres et des tailles importantes. La méthode utilisée ici cherchant à «combler des trous» nous donne de nombreux résultats optimaux comme (entre autre) la solution des 21 carrés.

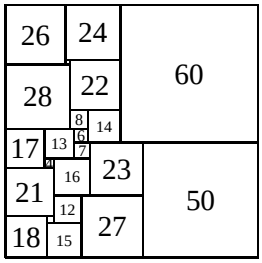
5.5 Quelques solutions

Pour des ordres allant de 21 à 135, nous donnons ici la première solution trouvée parmi les solutions précédentes avec 3 plateaux, 4 plateaux ou 2 plateaux (dans cet ordre). Nous pouvons trouver parmi ces découpages, certains intéressants :

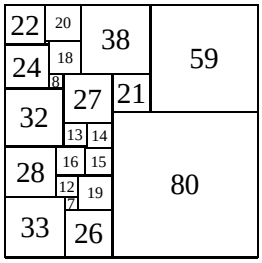
- le carré 112×112 d'ordre 21 qui est celui d'ordre minimum ;
- un carré 110×110 d'ordre 22 qui est celui de taille minimale et qui ne peut être trouvé avec 3 plateaux ;
- le carré 976×976 d'ordre 52 qui est une des rares solutions composées ;



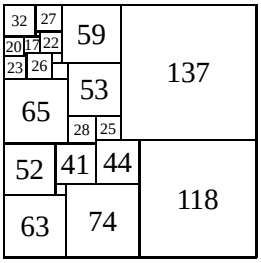
112×112, 21 Carrés



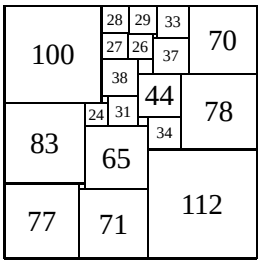
110×110, 22 Carrés



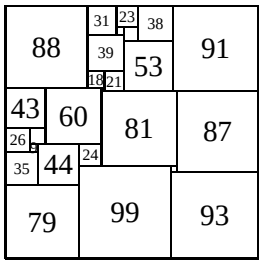
139×139, 23 Carrés



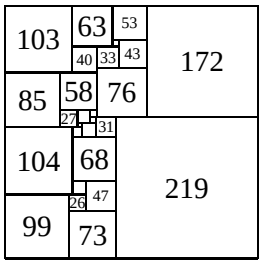
255×255, 24 Carrés



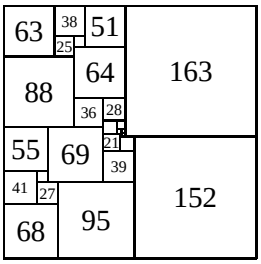
260×260, 25 Carrés



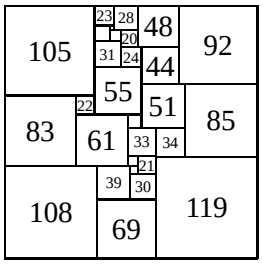
271×271, 26 Carrés



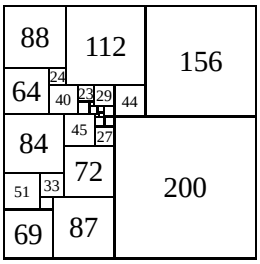
391×391, 27 Carrés



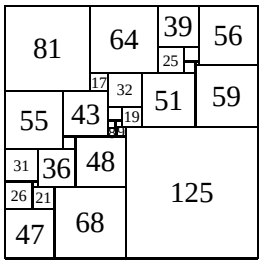
315×315, 28 Carrés



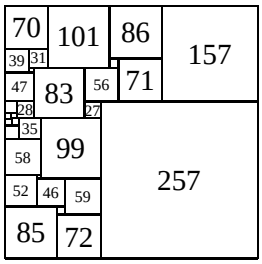
296×296, 29 Carrés



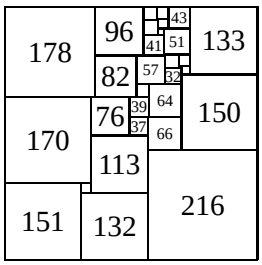
356×356, 30 Carrés



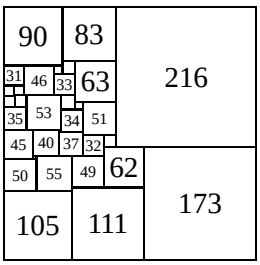
240×240, 31 Carrés



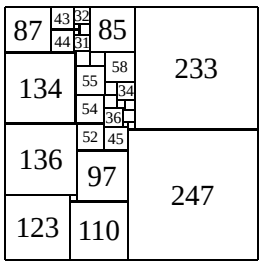
414×414, 32 Carrés



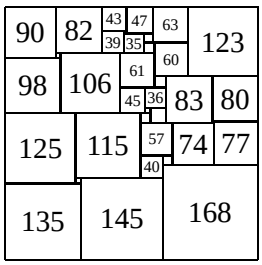
499×499, 33 Carrés



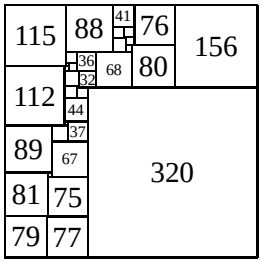
389×389, 34 Carrés



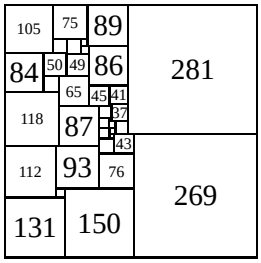
480×480, 35 Carrés



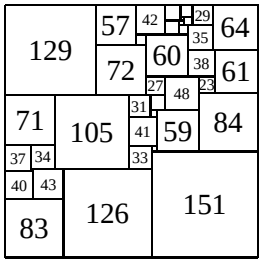
448×448, 36 Carrés



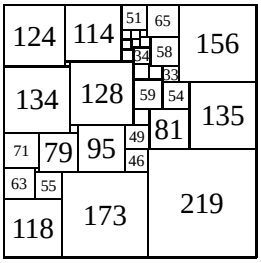
476×476, 37 Carrés



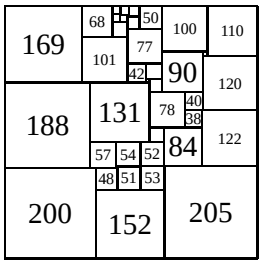
550×550, 38 Carrés



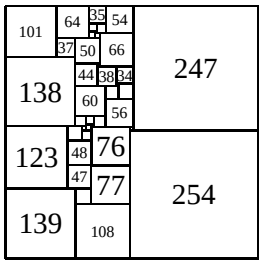
360×360, 39 Carrés



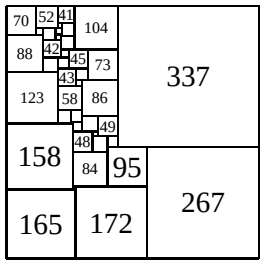
510×510, 40 Carrés



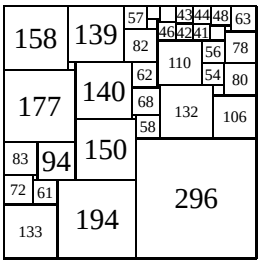
557×557, 41 Carrés



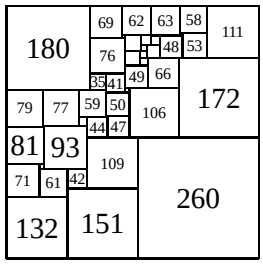
501×501, 42 Carrés



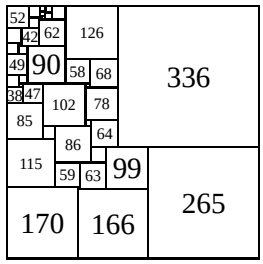
604×604, 43 Carrés



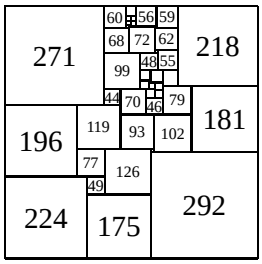
623×623, 44 Carrés



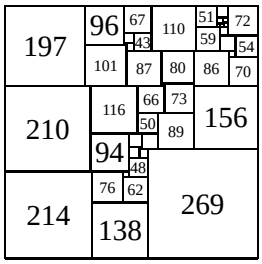
543×543, 45 Carrés



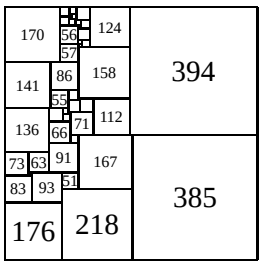
601×601, 46 Carrés



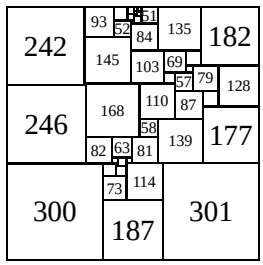
691×691, 47 Carrés



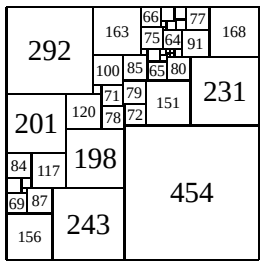
621×621, 48 Carrés



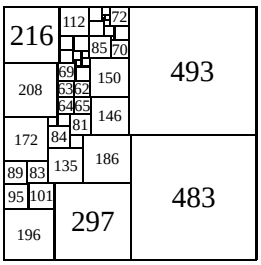
779×779, 49 Carrés



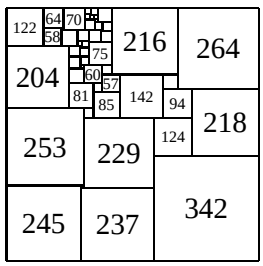
788×788, 50 Carrés



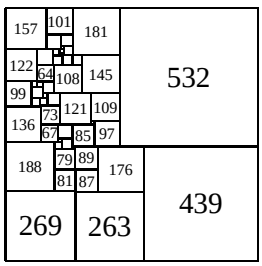
853×853, 51 Carrés



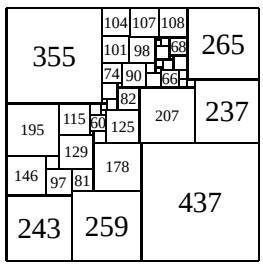
976×976, 52 Carrés



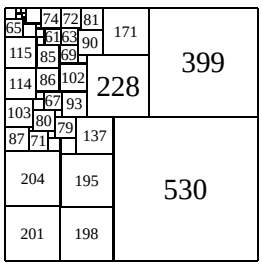
824×824, 53 Carrés



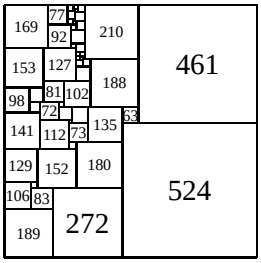
971×971, 54 Carrés



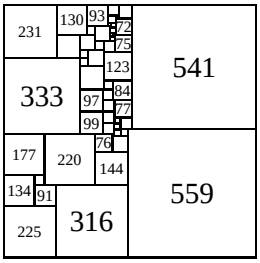
939×939, 55 Carrés



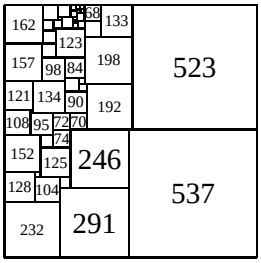
929×929, 56 Carrés



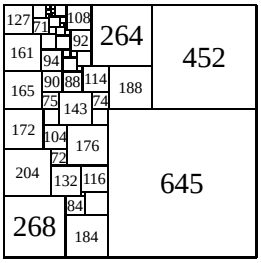
985×985, 57 Carrés



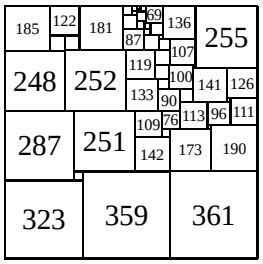
1100×1100, 58 Carrés



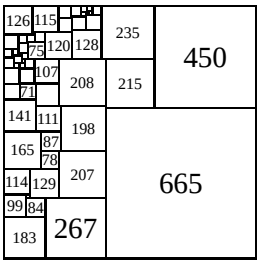
1060×1060, 59 Carrés



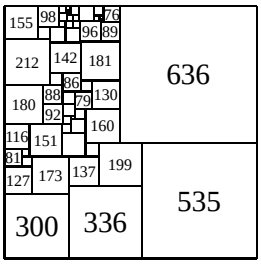
1097×1097, 60 Carrés



1043×1043, 61 Carrés

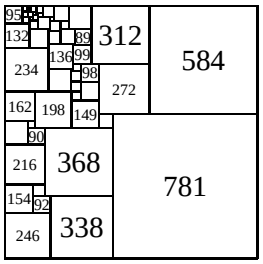


1115×1115, 62 Carrés

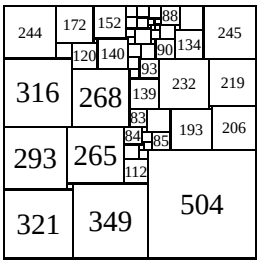


1171×1171, 63 Carrés

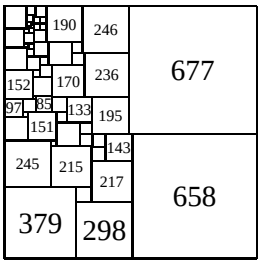
1263×1263, 64 Carrés



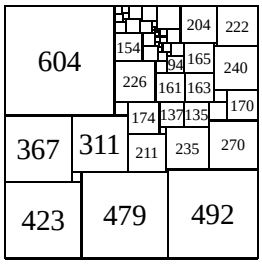
1365×1365, 65 Carrés



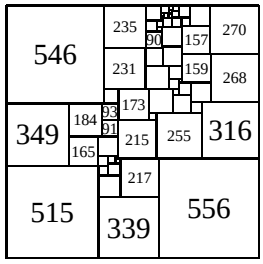
1174×1174, 66 Carrés



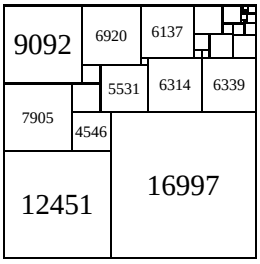
1335×1335, 67 Carrés



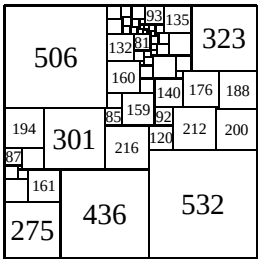
1394×1394, 68 Carrés



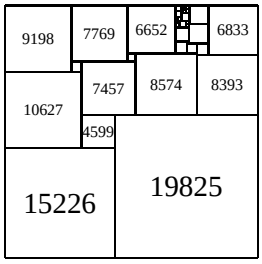
1410×1410, 69 Carrés



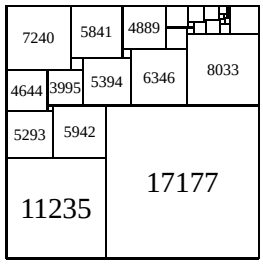
29448×29448, 70 Carrés



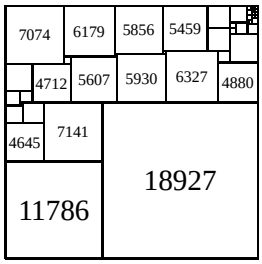
1243×1243, 71 Carrés



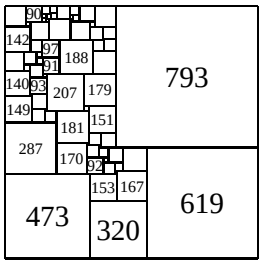
35051×35051, 72 Carrés



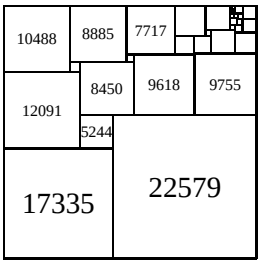
28412×28412, 73 Carrés



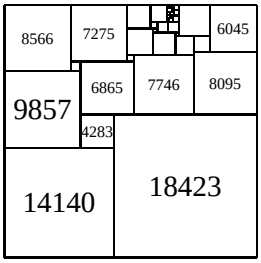
30713×30713, 74 Carrés



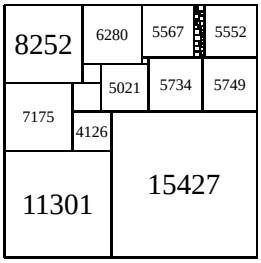
1412×1412, 75 Carrés



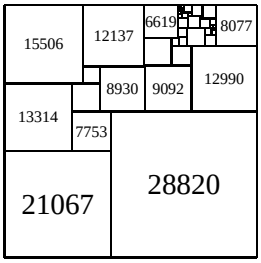
39914×39914, 76 Carrés



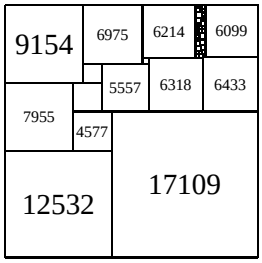
32563×32563, 77
Carrés



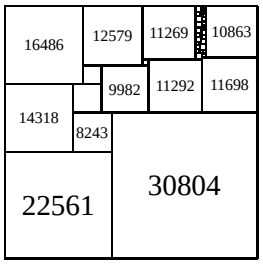
26728×26728, 78
Carrés



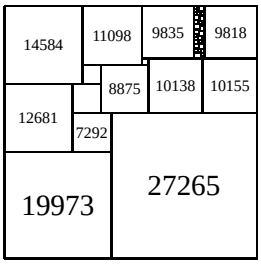
49887×49887, 79
Carrés



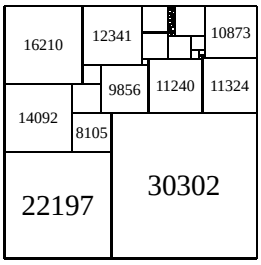
29641×29641, 80
Carrés



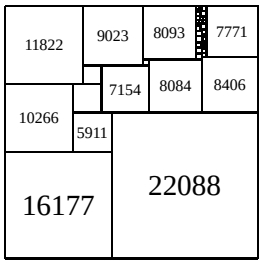
53365×53365, 81
Carrés



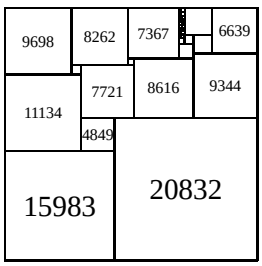
47238×47238, 82
Carrés



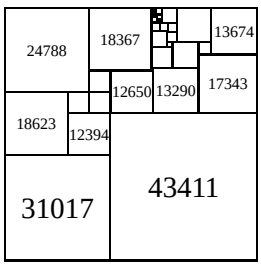
52499×52499, 83
Carrés



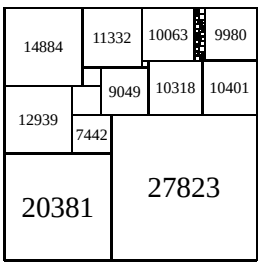
38265×38265, 84
Carrés



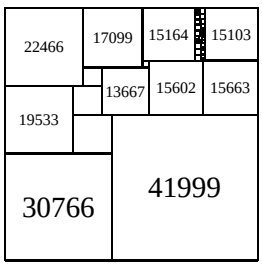
36815×36815, 85
Carrés



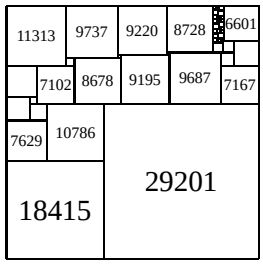
74428×74428, 86
Carrés



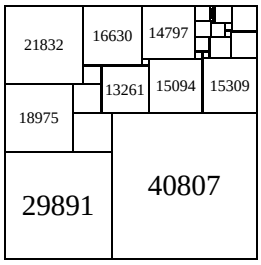
48204×48204, 87
Carrés



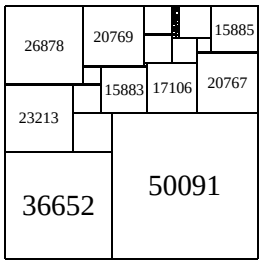
72765×72765, 88
Carrés



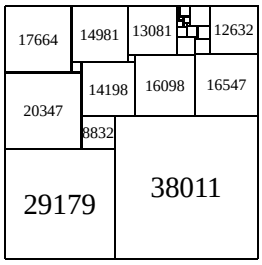
47616×47616, 89
Carrés



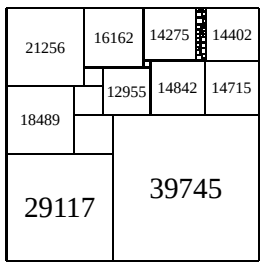
70698×70698, 90
Carrés



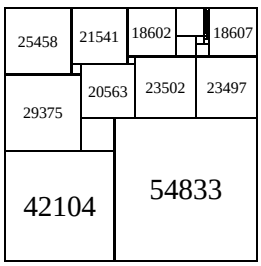
86743×86743, 91
Carrés



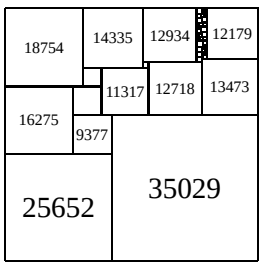
67190×67190, 92
Carrés



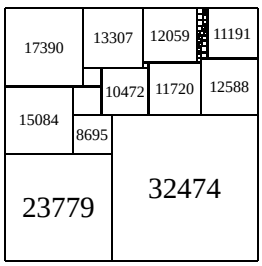
68862×68862, 93
Carrés



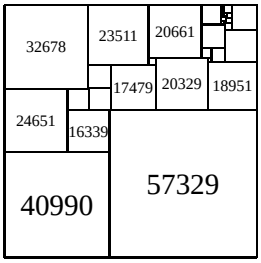
96937×96937, 94
Carrés



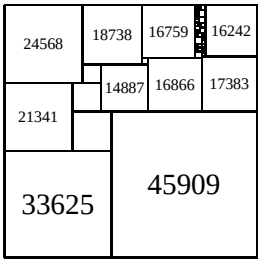
60681×60681, 95
Carrés



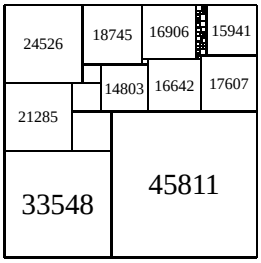
56253×56253, 96
Carrés



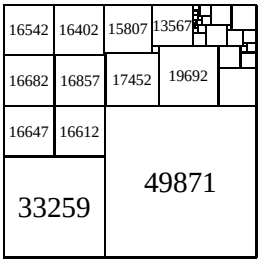
98319 × 98319, 97
Carrés



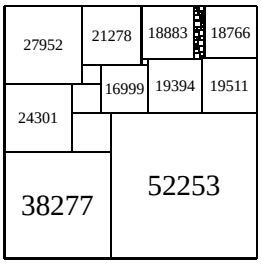
79534 × 79534, 98
Carrés



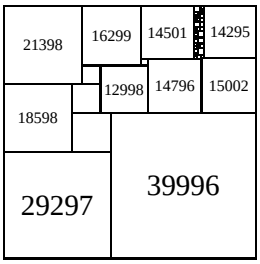
79359 × 79359, 99
Carrés



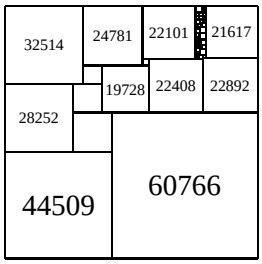
83130 × 83130, 100
Carrés



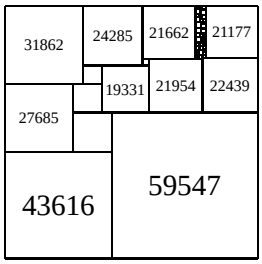
90530 × 90530, 101
Carrés



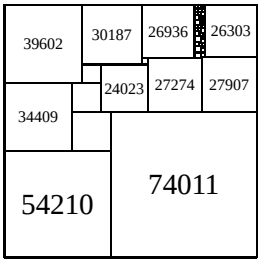
69293 × 69293, 102
Carrés



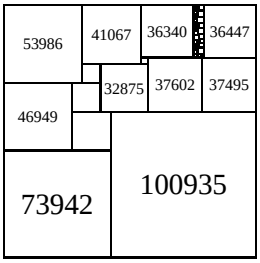
105275 × 105275, 103
Carrés



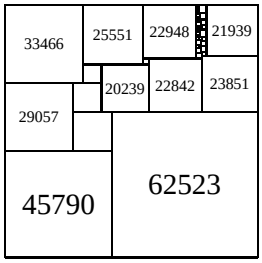
103163 × 103163, 104
Carrés



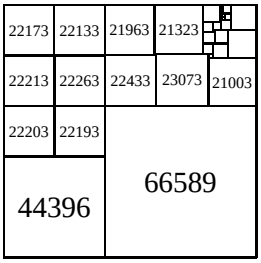
128221 × 128221, 105
Carrés



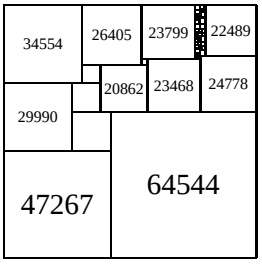
174877 × 174877, 106
Carrés



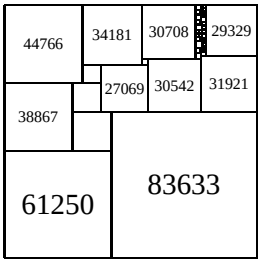
108313 × 108313, 107
Carrés



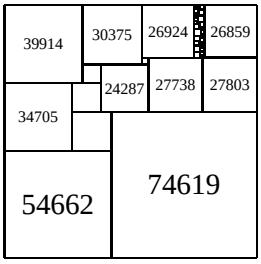
110985 × 110985, 108
Carrés



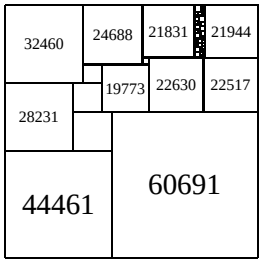
111811 × 111811, 109
Carrés



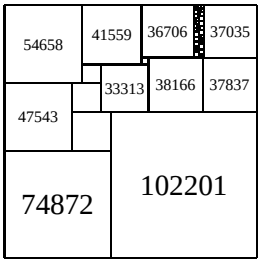
144883 × 144883, 110
Carrés



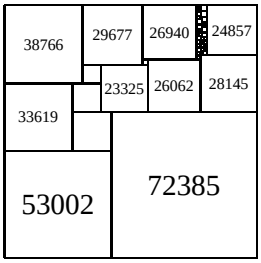
129281 × 129281, 111
Carrés



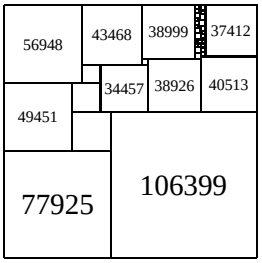
105152 × 105152, 112
Carrés



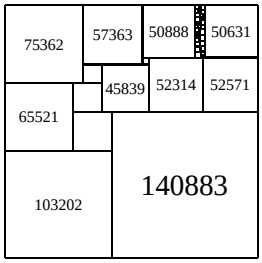
177073 × 177073, 113
Carrés



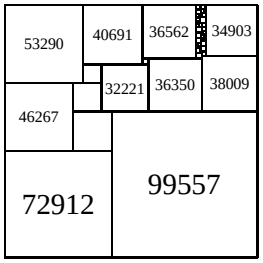
125387 × 125387, 114
Carrés



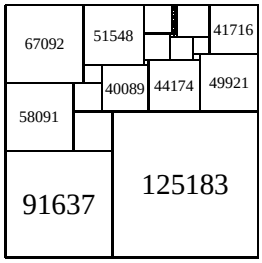
184324 × 184324, 115
Carrés



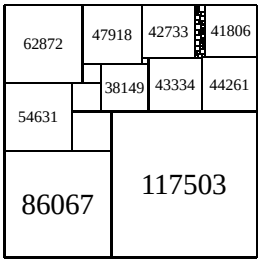
244085 × 244085, 116
Carrés



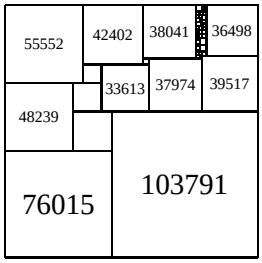
172469×172469, 117
Carrés



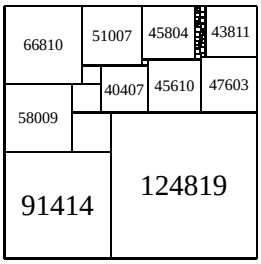
216820×216820, 118
Carrés



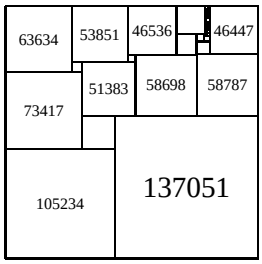
203570×203570, 119
Carrés



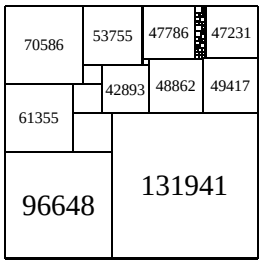
179806×179806, 120
Carrés



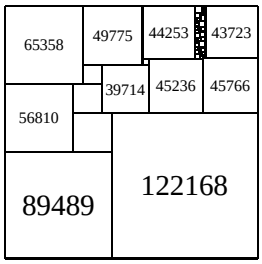
216233×216233, 121
Carrés



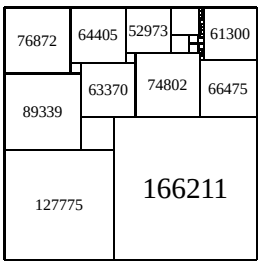
242285×242285, 122
Carrés



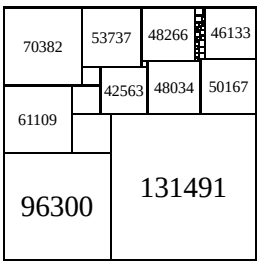
228589×228589, 123
Carrés



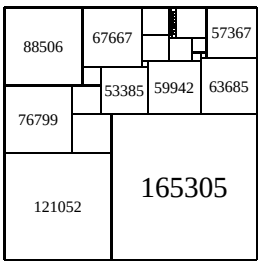
211657×211657, 124
Carrés



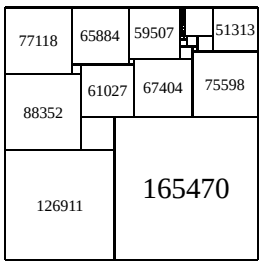
293986×293986, 125
Carrés



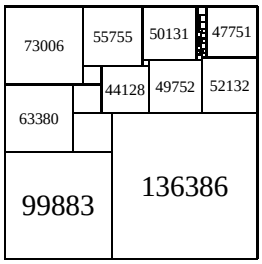
227791×227791, 126
Carrés



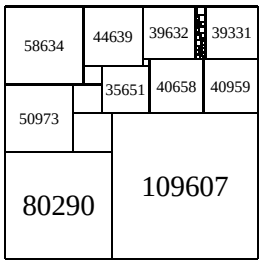
286357×286357, 128
Carrés



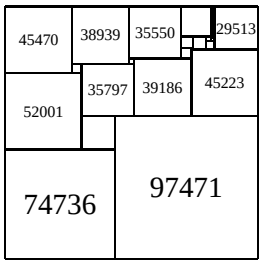
292381×292381, 130
Carrés



236269×236269, 131
Carrés



189897×189897, 132
Carrés



172207×172207, 135
Carrés

Chapitre 6

Problèmes connexes

Nous allons ici utiliser notre algorithme d'énumération sélective pour trouver des solutions à des problèmes adjacents. Il existe assez peu de résultats dans la littérature sur ce type de travaux mais Michael Goldberg [34] a travaillé sur des forme très originales de découpages.

6.1 Découpage parfait de cylindres

Dans notre problème initial, les éléments de la décomposition sont limités par les bords du carré à décomposer. Si on replie le bord droit sur le bord gauche pour former un cylindre, on peut alors s'affranchir de la contrainte des bords droit et gauche. Cette modification revient à autoriser un élément de la décomposition à dépasser sur la droite et à revenir sur la gauche.

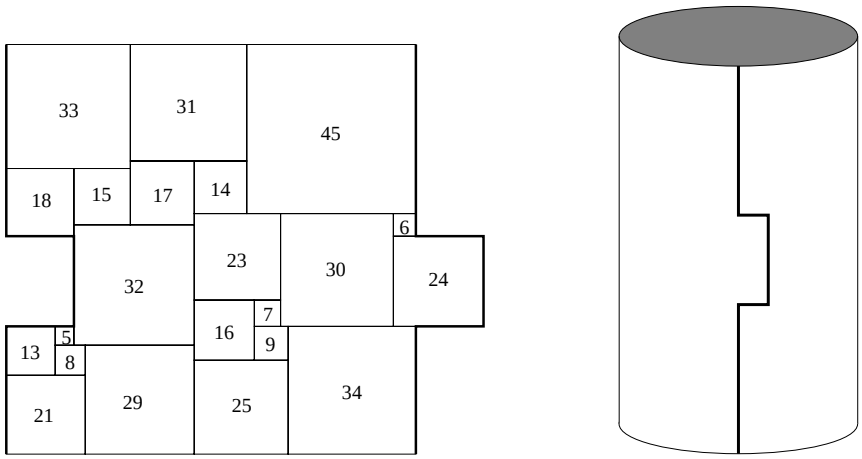


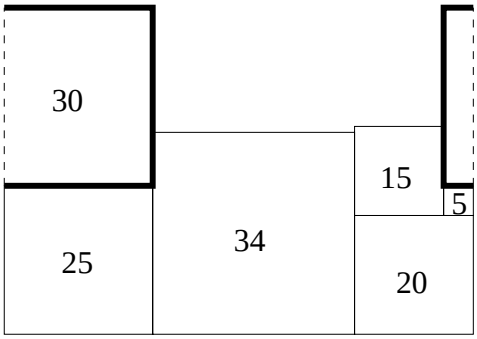
FIG. 6.1 – Découpage d'un cylindre «carré» $22 : 109 \times 109$

La figure 6.1 nous donne un exemple de découpage parfait de cylindres «carrés». Nous appellerons cylindre «carré» un cylindre fabriqué en repliant les bords d'un carré, c'est-à-dire, dont la hauteur du cylindre est égale à sa circonférence. Sur cette même idée, un élément de la décomposition (initialement un carré) est formé ici de 2 hauteurs et de 2 arcs de cercles égaux.

Dans [3], A. Augusteijn et A.J.W. Duijvestijn on trouvé les deux cylindres «carrés» d'ordres minimaux ($20 : 79 \times 79$ et $20 : 81 \times 81$). Leur méthode consiste à effectuer une énumération de graphes représentant des décompositions de cylindres et à résoudre les systèmes linéaires associés.

6.1.1 Algorithme

L'algorithme que nous allons utiliser pour résoudre ce problème est issu de la méthode d'énumération sélective que nous avons précédemment décrite. Dans le cas du découpage d'un cylindre, il est assez simple de voir les modifications à apporter à notre programme. Puisque nous utilisions, comme structure de donnée, une liste de plateaux pour décrire le dessus de la construction, la modification essentielle est de rendre cette liste circulaire, ainsi il sera possible de placer de nouveaux carrés comme suit :



6.1.2 Résultats

Tout comme pour l'algorithme initial d'énumération sélective, les résultats sont fonction du nombre de carrés placés à la base de la décomposition. Nous avons choisi une base avec 3 carrés car c'est la configuration qui nous donne les meilleurs résultats.

Avec une base de 3 carrés, en énumérant la taille de la base entre 6 et 500 nous obtenons les résultats suivants :

– 107 cylindres «carrés» et 4 623 cylindres «rectangulaires» ;
– le calcul a pris 2 heures 48 minutes sur un Pentium Pro 200 sous Linux.

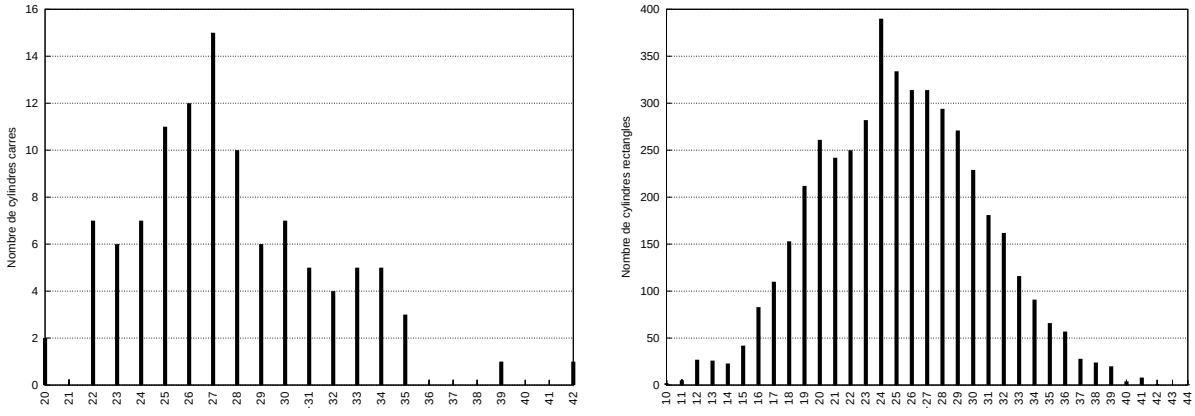


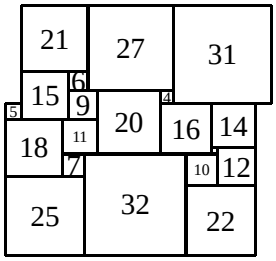
FIG. 6.2 – Répartition des décompositions de cylindres en fonction du nombre de carrés utilisés

6.1.3 Les principales solutions

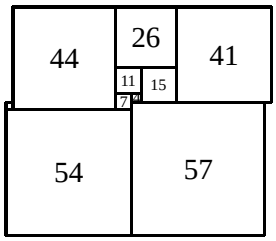
Nous avons placé ici les décompositions intéressantes de cylindres en carrés trouvées par notre algorithme :

1. le cylindre «carré» d'ordre et de taille minimale que nous ayons trouvé ;

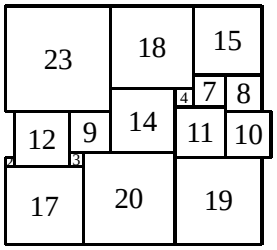
2. le cylindre «rectangulaire» d'ordre minimal trouvé ;
3. le cylindre «rectangulaire» de taille minimale trouvé.



79×79, 20 Carrés



111×98, 10 Carrés



56×52, 16 Carrés

6.2 Découpage parfait de tores

Ce que cherchons à faire ici est d'enlever la deuxième contrainte sur les bords. En autorisant les éléments à dépasser sur les côtés droit et gauche et sur les côtés haut et bas, on effectue en réalité le découpage d'un «tore». Les éléments d'un tel découpage sont alors définis par 4 arcs de cercles d'angles égaux.

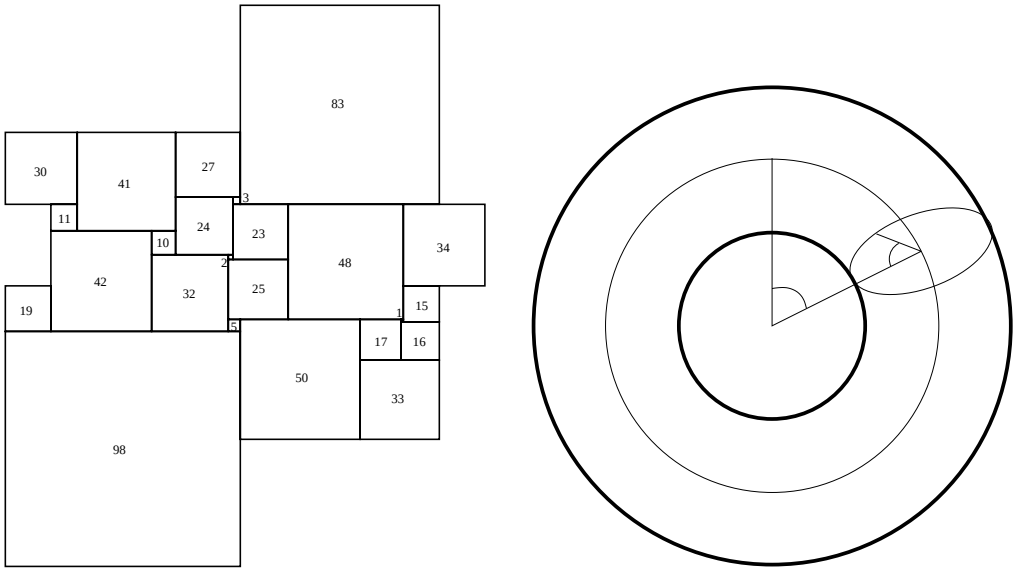


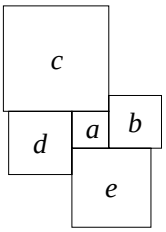
FIG. 6.3 – Découpage d'un tore «carré» 24 : 181×181

6.2.1 Première approche

Pour le découpage d'un tore, nous avons un problème car nous n'avons plus de base pour l'énumération initiale de notre algorithme. En effet, dans ce cas nous n'avons plus aucun repère pour la construction.

Une première approche se base sur la propriété 3 qui nous indique que le plus petit carré d'une décomposition ne peut pas se trouver sur un plateau, ce qui engendre une des deux configurations suivantes :

Ce qui est gênant dans nos algorithme ce sont les 8 symétries du problème. Si on prend comme repère le plus petit carré d'une décomposition il est possible de supprimer ces symétries en plaçant les contraintes suivantes :



- le carré de taille a est le plus petit de toute la décomposition ;
- le carré de taille b est à droite du carré a et sont alignés en bas (supprime une symétrie) ;
- le carré de taille b est plus petit que les carrés de tailles c, d, e (supprime les 2 autres symétries).

On peut ainsi prendre cette figure comme base de notre algorithme d'énumération. L'avantage étant que partant du carré le plus petit, on sait au départ que tous les carrés énumérés doivent être plus grands. De plus, une fois qu'on a choisi le carré de taille b, on sait que les carrés c, d, e sont plus grands.

Cette méthode semble donner une bonne base pour une énumération mais ici la construction ne s'effectue pas de bas en haut mais dans toutes les directions. Le problème est de savoir quelle est la taille limite d'un carré, quelle peut être sa taille maximale.

Les différentes méthodes que nous avons appliquées pour cet algorithme n'ont donné aucun résultat à cause de la complexité du traitement lors du placement d'un nouveau carré, pourtant cet algorithme a tourné pendant de nombreux jours...

6.2.2 Algorithme

Nous voulions à tout prix trouver au moins une solution de découpage d'un tore. Puisque nous avons un algorithme grandement efficace sur les carrés ainsi que sur la décomposition de cylindres, nous avons encore une fois modifié celui-ci pour le découpage de tores. La base de l'algorithme est identique à celui du découpage de cylindres.

Nous partons avec une base de 3 carrés, sauf qu'ici, nous avons décalé le dernier carré verticalement (voir figure 6.4).

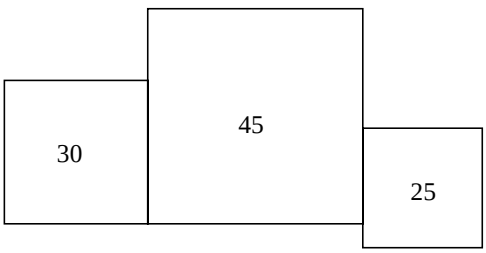


FIG. 6.4 – Enumération initiale pour le découpage d'un tore

En procédant ainsi, nous obtenons une base sur laquelle notre algorithme d'énumération

sélective peut opérer mais avec une légère modification quant à la limite supérieure pour la taille des carrés. En effet, la hauteur n'est plus limitée par un seul segment de droite horizontal mais par 2 segments (figure 6.5), ce qui complexifie les tests lors du placement d'un nouveau carré.

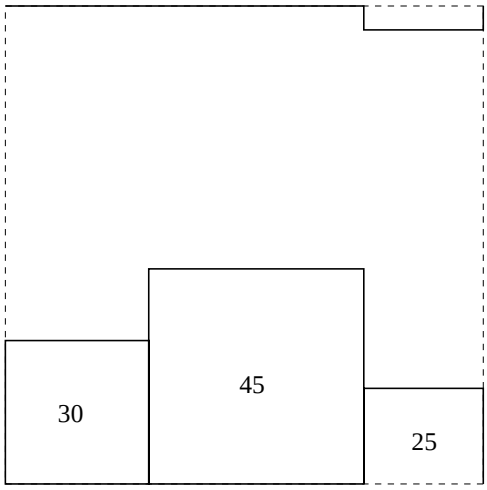


FIG. 6.5 – Délimitation de la surface à décomposer

Il est bien sûr possible d'utiliser plus de 3 carrés pour la base et de décaler plusieurs carrés verticalement mais dans ce cas on arrive dans une impasse due à un nombre trop grand de combinaisons.

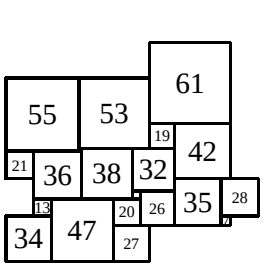
6.2.3 Résultats

Les résultats obtenus avec cet algorithme sont les suivants pour une énumération de la base de 6 jusqu'à 450 :

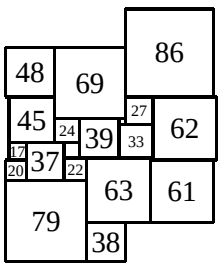
- Durée du calcul : 15 jours et 8 heures sur un Pentium Pro 200 sous Linux ;
- 1 tore «carré» et 35 tores «rectangulaires»

6.2.4 Principales solutions

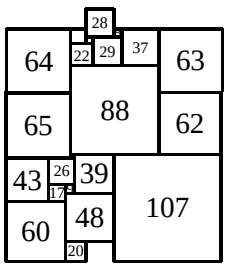
Nous avons mis ici toutes les solutions trouvées et la seule solution «carrée» est celle d'ordre 24 et de taille 181×181 (figure 6.6).



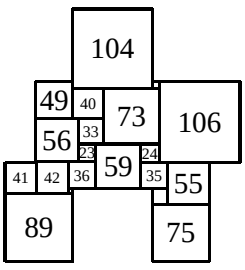
169×138, 20 Carrés



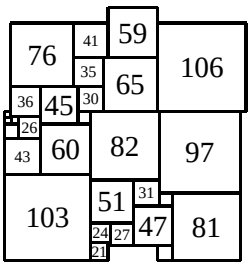
203×209, 20 Carrés



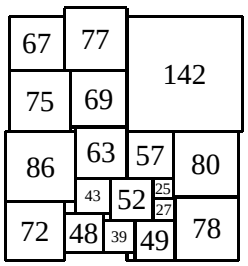
215×232, 23 Carrés



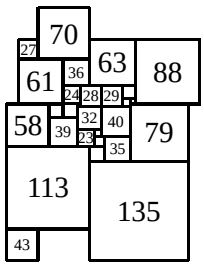
268×236, 21 Carrés



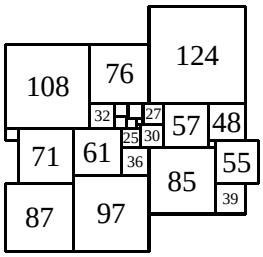
282×284, 31 Carrés



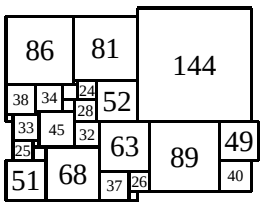
286×300, 23 Carrés



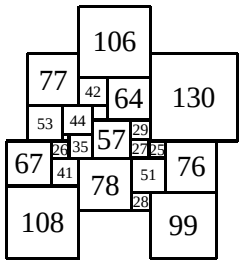
248×302, 28 Carrés



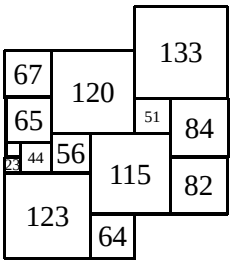
308×266, 26 Carrés



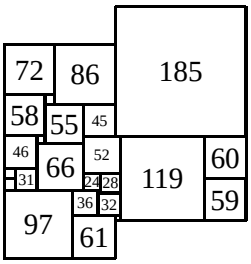
311×233, 28 Carrés



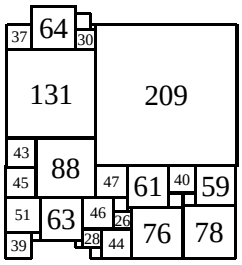
313×305, 25 Carrés



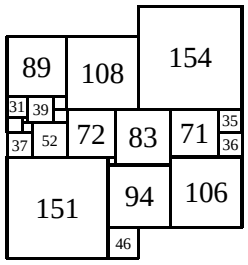
320×299, 15 Carrés



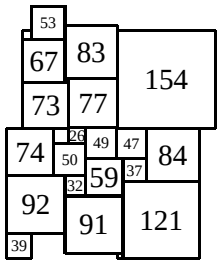
343×304, 26 Carrés



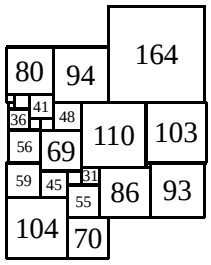
340×346, 31 Carrés



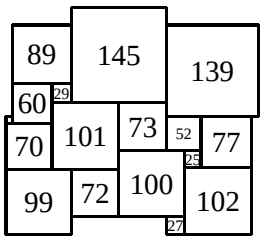
351×331, 23 Carrés



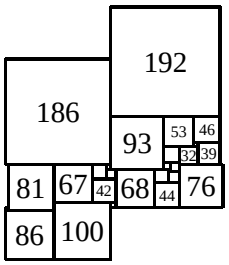
304×359, 24 Carrés



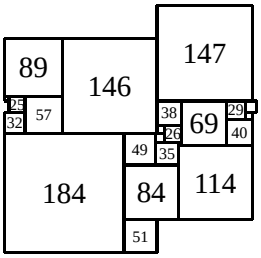
338×360, 25 Carrés



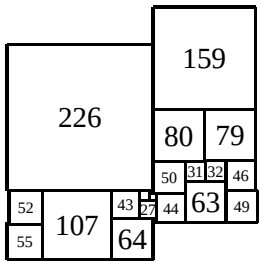
373×318, 17 Carrés



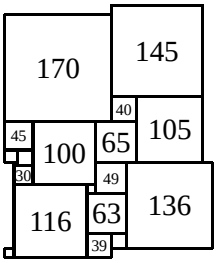
378×353, 27 Carrés



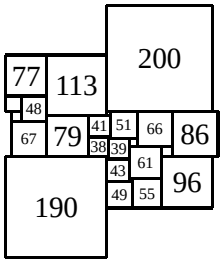
382×330, 23 Carrés



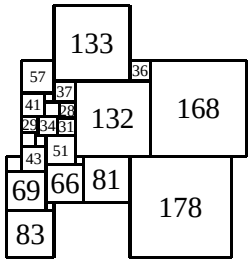
385×333, 23 Carrés



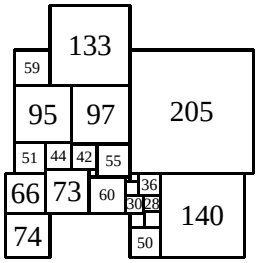
315×386, 19 Carrés



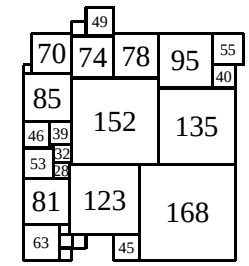
390×382, 26 Carrés



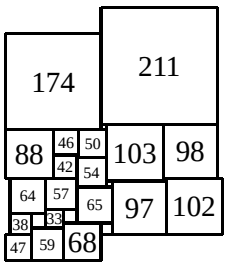
394×346, 27 Carrés



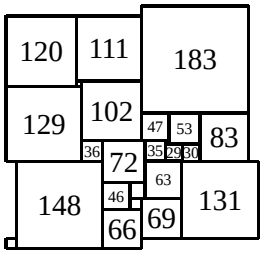
397×345, 25 Carrés



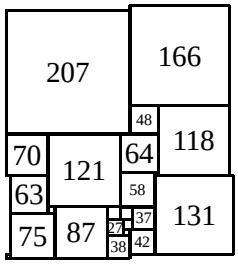
372×398, 29 Carrés



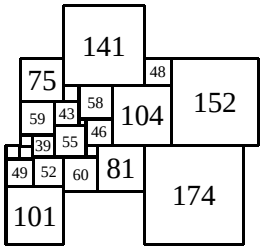
385×411, 26 Carrés



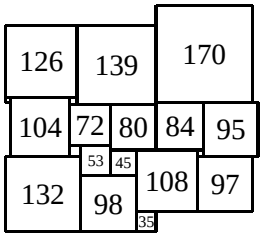
414×397, 25 Carrés



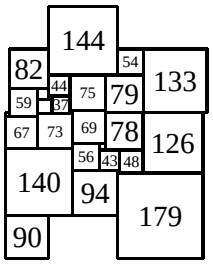
373×415, 23 Carrés



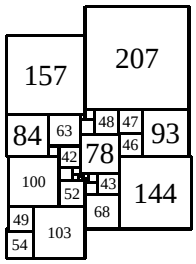
416×326, 25 Carrés



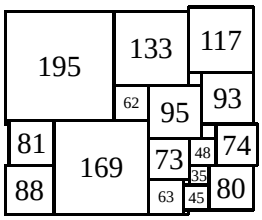
435×362, 19 Carrés



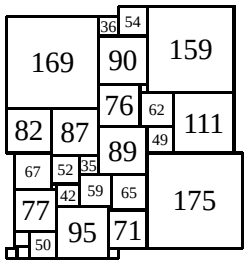
413×438, 28 Carrés



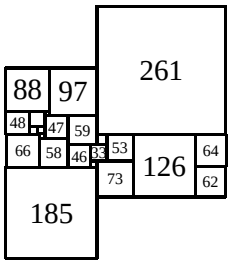
364×444, 35 Carrés



445×364, 21 Carrés



418×445, 32 Carrés



446×387, 25 Carrés

6.3 Décomposition d'un cube en cubes de tailles distinctes

On peut se poser la question de savoir si notre problème peut être étendu en 3 dimensions. La question que l'on se pose alors est de savoir s'il est possible de décomposer un cube en plusieurs cubes de tailles distinctes.

Supposons qu'il existe une telle décomposition et soient $C_1^0, ..., C_{m_0}^0$ les cubes situés sur la face inférieure. Soit $C_{i_0}^0$ le plus petit de ces cubes. La projection de cette face étant la décomposition d'un carré en carrés distincts et ayant vu (propriété 4) que le plus petit carré ne pouvait pas se trouver sur le bord, le plus petit cube $C_{i_0}^0$ ne peut pas se trouver sur le bord de la face inférieure. Le cube $C_{i_0}^0$ est donc entouré de cubes de plus grande tailles.

Prenons maintenant les cubes $C_1^1, ..., C_{m_1}^1$ situés au dessus de $C_{i_0}^0$, qui existent forcément puisque les carrés autour de $C_{i_0}^0$ sont plus grands. On peut, de la même manière que précédemment, prendre le cube le plus petit $C_{i_1}^1$ et recommencer notre raisonnement à l'infini.

Il est donc impossible de décomposer un cube en un nombre fini de cubes de tailles distinctes et cette démonstration peut être étendue à un espace de degré $n > 3$.

Cette démonstration est aussi démontrée dans [13] et [37].

Robert J.MacG. Dawson montre par ailleurs dans [18, 19] qu'il est également impossible de paver l'espace avec des cubes de tailles distinctes.

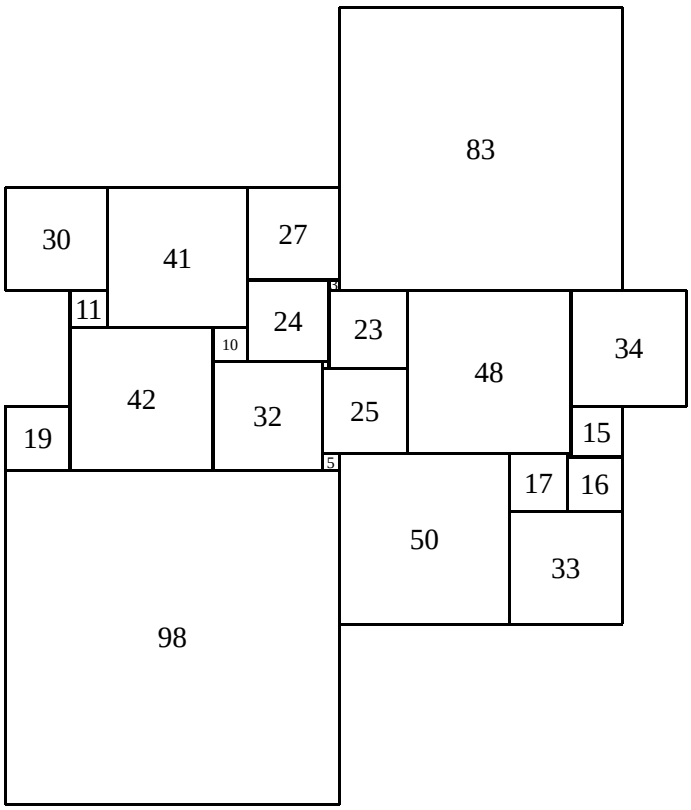


FIG. 6.6 – Décomposition d'un tore «carré» en carrés distincts ($24 : 181 \times 181$)

Chapitre 7

Conclusion

On trouve dans la littérature de nombreux articles sur notre sujet. La plupart portent sur l'énumération de graphes schématisant les décompositions de carrés en carrés. Quelques uns portent sur le calcul de nouvelles décompositions par combinaisons et modifications de décompositions connues.

Une petite contribution fut l'introduction d'une méthode naïve d'énumération de graphes qui n'évite pas toutes les redondances mais qui s'est révélée très rapide.

Une autre contribution fut de tester plusieurs algorithmes pour résoudre en précision infinie les systèmes linéaires associés à ces graphes et de choisir l'algorithme le plus approprié. Il nous a été possible de calculer toutes les décompositions parfaites de carrés pour des ordres n allant de 21 à 26. Pour $n = 26$ nous mettons 16 jours sur une machine datant de janvier 1997. Avec les machines actuelles, nous estimons pouvoir atteindre $n = 28$ en deux mois, alors que la limite connue est $n = 26$ (C.J. Bouwkamp, A.J.W. Duijvestijn, décembre 1994).

Notre première contribution originale fut l'étude de configurations de carrés n'apparaissant jamais dans une décomposition. Cette étude et le fait que les machines actuelles tournent infiniment plus vite, nous a permis d'aborder le calcul exhaustif des décompositions entières d'un carré de taille donnée n. Nous avons dû cependant nous limiter à des tailles n relativement faibles.

Notre seconde contribution originale fut une méthode sélective (non exhaustive) d'énumération des décompositions entières de carrés qui retrouve les solutions bien connues de petites tailles mais en donne d'autres de tailles très grandes. En modifiant cette méthode nous avons aussi trouvé des décompositions originales de cylindres et de tores. Il est à noter que la décomposition d'un tore peut être vue comme un pavage régulier du plan. Bien entendu, il en va de même pour la décomposition d'un cylindre ou tout simplement d'un carré.

Nous avons essayé d'autres méthodes, notamment par résolution de contraintes discrètes, mais sans résultat. Voici une méthode que nous n'avons pas essayé : on énumère des égalités de surfaces de la forme $n^2 = \Sigma_{i=1}^{k} n_i^2$, avec les n_i tous distincts, et par résolution de contraintes discrètes, (ici ça fonctionne bien) on vérifie s'il est possible de décomposer un carré de taille entière connue n en carrés de tailles entières connues n_i.

Terminons par la belle égalité de surfaces qui malheureusement ne donne lieu à aucune décomposition de carré :

$$70^2 = 1^2 + 2^2 + \cdots + 23^2 + 24^2$$

Bibliographie

[1] A. Aggoun and N. Beldiceanu, Extending CHIP to solve complex scheduling and packing problem, *Journées Francophones de Programmation Logique*, 1992.

[2] P.J. van Albada, La dissection du carré en carrés, *Bulletin de la Société Mathématique de Belgique*, Vol. 20, Pages 161-170, 1968.

[3] A. Augusteijn and A.J.W. Duijvestijn, Simple Perfect Square-Cylinders of Low Order, *Journal of combinatorial theory*, Series B35, pages 333-337, 1983.

[4] David-Olivier Azulay et Jean-François Pique, Optimisation des Q-matrices pour la résolution de contraintes linéaires, *Journées Francophones de Programmation Logique et programmation par Contraintes*, Pages 199-212, mai 1998.

[5] David-Olivier Azulay, Programmation linéaire exacte avec Q-matrices, *Thèse de doctorat*, Faculté des sciences de Luminy, Université de la Méditerranée, 1999.

[6] C.J. Bouwkamp, On the dissection of rectangles into squares, *Proc. Kon. Ned. Akad. Wetensch. Amsterdam* **49** (1946), 1176-1188 ; **50** (1947), 58-71, 72-78.

[7] C.J. Bouwkamp, A.J.W. Duijvestijn et P. Medema, Catalogue of simple squared rectangles of order nine through fourteen and their elements, *Technische Hogeschool*, Eindhoven, The Netherlands, 50 Pages, May 1960.

[8] C.J. Bouwkamp, A.J.W. Duijvestijn et J. Haubich, Catalogue of simple perfect squared rectangles of order 9 through 18, Philips Research Laboratories, Eindhoven, The Netherlands, Vols 1-12, 3090 Pages, 1964.

[9] C.J. Bouwkamp, On some new simple perfect squared sqaures, *Discrete Mathematics* 106-7, Pages 67-75, 1992.

[10] C.J. Bouwkamp et A.J.W. Duijvestijn, Catalogue of simple perfect squared squares of order 21 through 25, EUT Report 92-WSK-03, Eindhoven, The Netherlands, 207 Pages, 1992.

[11] C.J. Bouwkamp et A.J.W. Duijvestijn, Album of simple perfect squared squares of order 26, EUT Report 94-WSK-02, Eindhoven University of Technology, The Netherlands, Décembre 1994.

[12] C.J. Bouwkamp, On step-2 transforms for simple perfect squared squares, *Discrete Mathematics* 179, Pages 243-252, 1998.

[13] R.L. Brooks, C.A.B. Smith, A.H. Stone and W.T. Tutte, The dissection of rectangles into squares, *Duke Math. J.* **7** (1940) 312-340.

[14] R.L. Brooks, A Procedure for Dissecting a Rectangle into Squares, and an Example for the Rectangle Whose Sides Are in the Ratio 2 :1, *Journal of Combinatorial Theory 8*, pages 232-243, 1970.

[15] R.L. Brooks, C.A.B. Smith, A.H. Stone and W.T. Tutte, Determinants and current flows in electric networks, *Discrete Mathematics* 100, Pages 291-301, 1992.

[16] Alain Colmerauer, An Introduction to Prolog III, *CACM*, pages 25-30, 28(4) :412-418, 1990.

[17] Yves Colombani, Carrés Parfaits et Contraintes sur Intervalles d'Entiers, *Mémoire de D.E.A.*, Faculté des Sciences de Luminy, Université de la Méditerranée, 1993.

[18] Robert J.MacG. Dawson, On filling space with different integer cubes, *Journal of combinatorial theory*, Serie A 36, Pages 221-229, 1984.

[19] Robert J.MacG. Dawson, On the impossibility of packing space with different cubes, *Journal of Combinatorial Theory*, Serie A 48, Pages 174-188, 1988.

[20] Max Dehn, Über Zerlegung von Rechtecken in Rechtecke. *Math. Ann.* **57** (1903) 314-332.

[21] A.J.W. Duijvestijn, Electronic Computation of Squared Rectangles, Thesis, *Technological University, Eindhoven, The Netherlands*, 1962 ; *Philips Research Reports 17*, pages 523-612, 1962.

[22] A.J.W. Duijvestijn, Simple Perfect Squared Square of Lowest Order, *Journal of Combinatorial Theory*, Series B25, pages 240-243, 1978.

[23] A.J.W. Duijvestijn, A Lowest Order Simple Perfect 2×1 Squared Rectangle, *Journal of Combinatorial Theory*, Series B26, pages 372-374, 1979.

[24] A.J.W. Duijvestijn and P. Leeuw, Lowest order squared rectangles with the largest element not on the boundary, *Mathematics of Computation*, Volume 37, Number 155, Pages 223-228, 1981.

[25] A.J.W. Duijvestijn, P.J. Federico and P. Leeuw, Compound Perfect Squares, *American Math. Monthly*, vol 89, pages 15-32, 1982.

[26] A.J.W. Duijvestijn, Simple perfect squared squares and 2×1 squared rectangles of order 26, *Mathematics of Computation*, Volume 65, Number 215, Pages 1359-1364, July 1996.

[27] P.J. Federico, Some Simple Perfect 2×1 Rectangles, *Journal of Combinatorial Theory*, pages 244-246, 1970.

[28] P.J. Federico, Squaring rectangles and squares : A historical review with annotated bibliography, in *Graph Theory and Related Topics*, Academic Press, pages 173-196, 1979.

[29] Ian Gambini, A method for cutting squares into distinct squares, *Discrete Applied Mathematics*, vol 98, pp. 65-80, nov 1999.

[30] Martin Gardner, How rectangles, including squares, can be divised into squares of unequal size, *Scientific American*, Vol. 199, Pages 136-142, 1958.

[31] Martin Gardner, *The second scientific american book of mathematical puzzles and diversions*, pp. 186-209, 250. Simon and Schuster, New York 1961.

[32] Keith O. Geddes, Stephen R. Czapor, George Labahn, *Algorithms for computer algebra*, Kluwer Academic Publishers, 1992

[33] Michael Goldberg and W.T. Tutte, Solution of problem E401, *American Mathematical Monthly*, vol 47, Pages 570-572, Oct. 1940.

[34] Michael Goldberg, The squaring of developable surfaces, *Scripta Math.* 18, Pages 17-24, 1952.

[35] B. Grünbaum and G. Shephard, *Tilings and patterns*, pp 76-81. W.H. Freeman And Co., N.Y., 1987.

[36] Pascal van Hentenryck, Vijay Saraswat et Yves Deville, Design, Implementation and Evaluation of the Constraint Language `cc(fd)`, *Lecture Notes in Computer Science*, vol 910, Pages 293-316, 1994.

[37] Ross Honsberger, Squaring the square, *Ingenuity in Mathematics, New Mathematical Library*, No.23, Pages 46-60, Random House, New York ; now by Math. Assoc. Amer., Washington, D.C., 1970.

[38] N.D. Kazarinoff and R. Weitzenkamp, Squaring rectangles and squares, *Amer. Math. Monthly,* vol. 80, Pages 877-888, 1973.

[39] M. Kraitchik, La mathématique des jeux ou récréations mathématiques, *Stevens Frères*, Bruxelles, Page 272, 1930.

[40] Zbigniew Moron, O rozkladach prostokatow na kwadraty (en Polonais : Sur la dissection de rectangles en carrés), *Przeglad Mat. Fiz. 3,* Pages 152-153, 1925.

[41] H. Reichardt et H. Toepken, Solution of problem 271, *Jber. Deutsch. Math. Verein*, 50, Part 2, Pages 13-14, 1940.

[42] J.D. Skinner II, Uniquely squared squares of a common reduced side and order, *Journal of combinatorial theory,* Series B 54, Pages 155-156, 1992.

[43] J.D. Skinner II, *Squared Squares : Who's Who & What's What*. Published by the author, 1993.

[44] R. Sprague, Beispiel einer Zerlegung des Quadrats in lauter verschiedene Quadrate, *Math. Z. 45,* pages 607-608, 1939.

[45] Ian Stewart, Squaring the Square, *Scientific American*, pages 74-76, July 1997.

[46] W.T. Tutte, Squared rectangles, *Proceedings of the IBM Scientific Symposium on Combinatorial Problems,* pages 3-9, 16-18 March 1964.

[47] W.T. Tutte, The Quest of the Perfect Square, *American Math. Monthly 72*, pages 29-35, 1965.

[48] W.T. Tutte, Graph Theory As I Have Known It, *Oxford Lecture Series in Mathematics and Its Applications*, 1998

[49] T.H. Willcocks, Problem 7795 and solution, *Fairy Chess Rev. 7,* 1948.

[50] J.C. Wilson, A Method for Finding Simple Perfect Squared Squarings, Thesis, *University of Waterloo*, 1967.

[51] Jianyang Zhou, Calcul de plus petits produits cartésiens d'intervalles : application au problème d'ordonnancement d'atelier, *Thèse de doctorat*, Faculté des sciences de Luminy, Université de la Méditerranée, 28 Mars 1997.