Résolution de puzzles dépourvus d'indice graphique

BEHAGUE Quentin

15 mars 2024

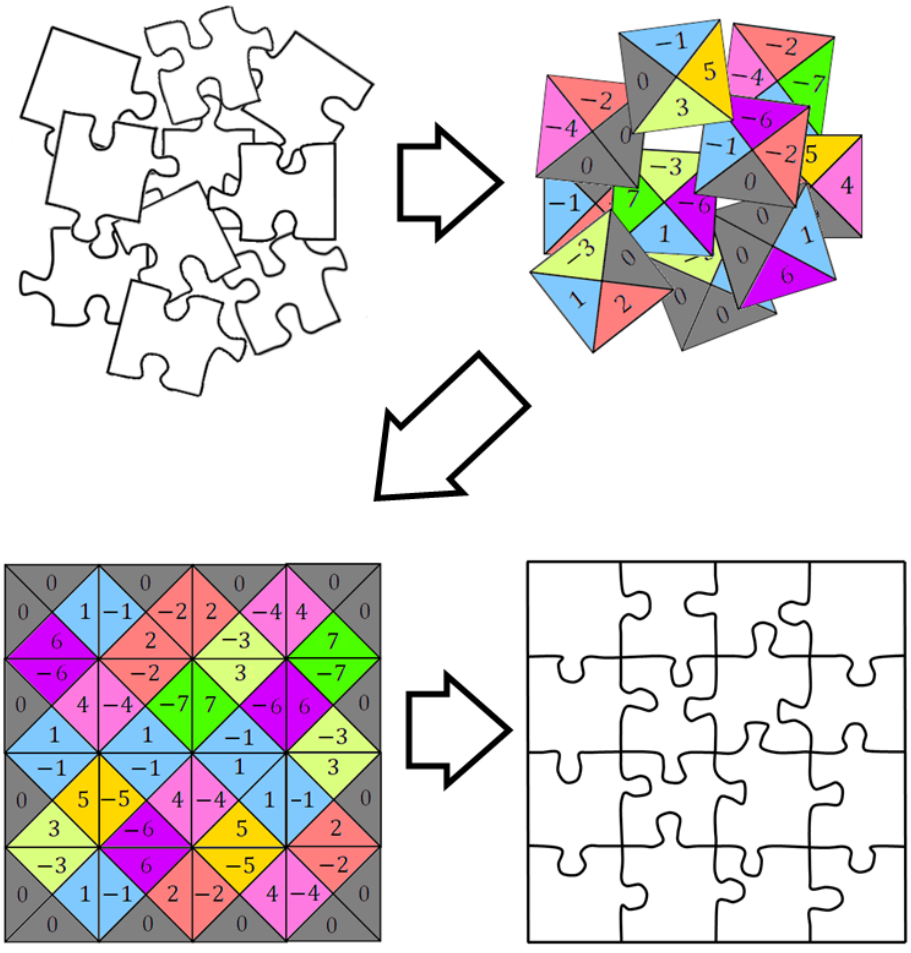

Table des matières

1 Introduction : **3**

2 Conversion du puzzle **3**
2.1 Encodage en chaîne . . . 3
2.2 Algorithme des k-moyennes . . . 4
2.3 Une seconde méthode . . . 4

3 Démonstration de NP-complétude **5**
3.1 Problème CONTOUR . . . 5
3.2 Problème PUZZLE . . . 6
3.3 Problème PUZZLE-ORIENTATION . . . 7
3.4 Problème PUZZLE-POSITION . . . 8

4 Problème PUZZLE-ORIENTATION **11**
4.1 α–approximation . . . 11
4.2 Algorithme exact . . . 15

5 Problème PUZZLE-POSITION **16**
5.1 α-approximation . . . 16
5.2 Algorithme efficace dans les cas pratiques . . . 17
5.2.1 Fonctionnement de l'algorithme : . . . 17
5.2.2 Terminaison et complexité . . . 18
5.2.3 Instances pour lesquelles l'algorithme ne fournit pas de solution . . . 18
5.2.4 Analyse dans le cas pratique : . . . 19

6 Algorithme exact pour le problème PUZZLE **20**
6.1 Backtracking sur les positions . . . 20
6.2 Conclusion . . . 20

7 Annexes **21**
7.1 Bibliographie . . . 21
7.2 Définition des problèmes . . . 21
7.3 Algorithmes . . . 22
7.3.1 Algorithme des k-moyennes . . . 22
7.3.2 $\frac{2}{k}$-approximation de k-Set packing . . . 23
7.3.3 Obtention des sous-ensembles de 4 pièces valides . . . 23

Liste des Algorithmes

1 Propagation des contraintes . . . 18
2 Algorithme des k-moyennes . . . 22
3 Algorithme pour k-Set packing . . . 23
4 Obtention des sous-ensembles de 4 pièces . . . 23

1 Introduction :

Introduits en 1760 comme un moyen ludique d'apprendre la géographie, les puzzles sont une activité populaire à la fois amusante et stimulante. Loin d'être uniquement un jeu, la résolution d'un puzzle représente un vrai défi algorithmique. Ce document traite le cas des puzzles monochromatiques.

La section 2 décrit la méthode employée pour convertir un tel puzzle en une instance du problème de décision PUZZLE. La section 3 établit la NP-complétude de ce problème, ainsi que celle de plusieurs de ses variantes. Les sections 4 et 5 introduisent le concept d'α-approximation et établissent des approximations des variantes PUZZLE-ORIENTATION et PUZZLE-POSITION respectivement, ainsi que des algorithmes fournissant des solutions exactes de ces deux problèmes. Enfin, la section 6 fournit un algorithme de résolution exact pour PUZZLE.

La définition de tous les problèmes traités est disponible en annexe (8.2). Un générateur d'instances des trois problèmes mentionnés ci-dessus est disponible à l'adresse suivante :

`https://behaguequentin.github.io/Programmes/Puzzle.html`

2 Conversion du puzzle

2.1 Encodage en chaîne

Étant donné que nous travaillons uniquement avec des puzzles monochromatiques, la seule information dont nous disposerons sera les courbes définissant les contours des pièces. Pour convertir cette information en une instance de PUZZLE, nous allons appliquer une version modifiée de la méthode d'encodage en chaîne décrite par Freeman et Garder [1].

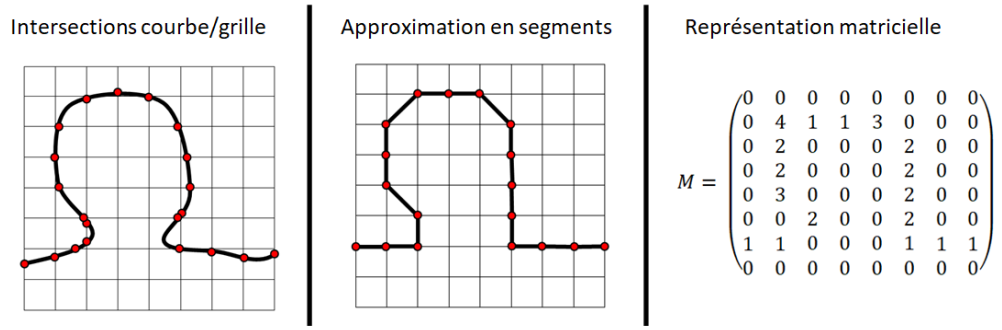

FIGURE 1 – Procédé de conversion matricielle d'un bord

Cette méthode fonctionne de la manière suivante :

— Chaque bord de pièces est placé sur une grille.

— Les intersections entre la courbe représentative du bord de la pièce et la grille sont enregistrées.

— Ces points sont alors approximés à l'intersection de la grille la plus proche.

— Chaque case de la grille contient alors un motif qui correspond à un entier :

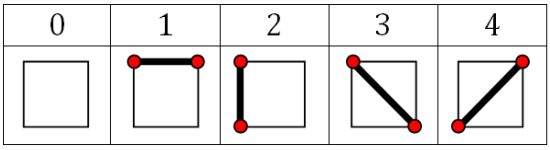

FIGURE 2 – Correspondances des motifs

2.2 Algorithme des k-moyennes

Pour obtenir une instance de PUZZLE, il faut maintenant regrouper les bords des pièces par similitudes. Pour cela, on utilise la méthode des k-moyennes. (Le fonctionnement de l'algorithme est détaillé en annexe (8.3).) L'avantage de cette méthode est qu'elle fonctionne aussi pour des entrées réelles, pour lesquelles les courbes seraient, par exemple, issus de scans de pièces réelles, ce qui impliquerait que deux courbes légèrement différentes puissent correspondre à des bords compatibles.

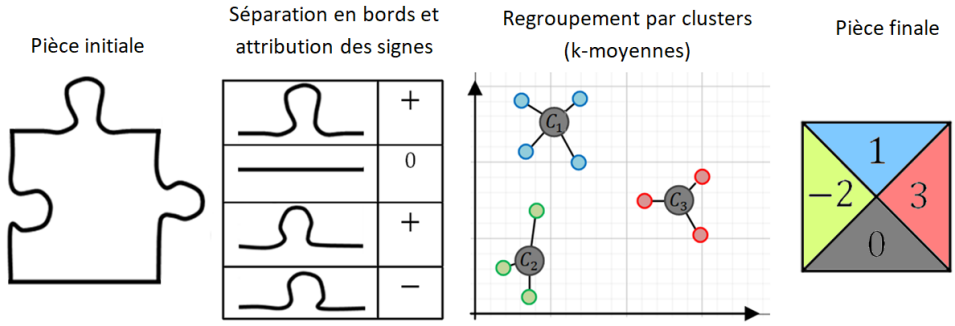

FIGURE 3 – Conversion en une instance de PUZZLE

En procédant ainsi, il est alors possible d'associer un entier à chaque bord. La valeur 0 est réservée aux bords plats. Les signes positifs et négatifs sont respectivement associés aux bords mâles et femelles.

2.3 Une seconde méthode

On remarque que la complexité temporelle et spatiale de la méthode de représentation des bords précédente suit une relation quadratique avec la précision que l'on utilise pour décrire la pièce. On peut donc considérer une méthode alternative qui permettra d'obtenir une complexité linéaire en la précision.

On va se servir de la forme particulière d'une pièce de puzzle : une abscisse ne peut être associée qu'à un maximum de trois ordonnées (voir figure 4). Il est alors possible de regarder les ordonnées associés à une subdivision de taille n de l'intervalle sur lequel est définie la courbe, et en déduire une matrice représentative de chaque bord.

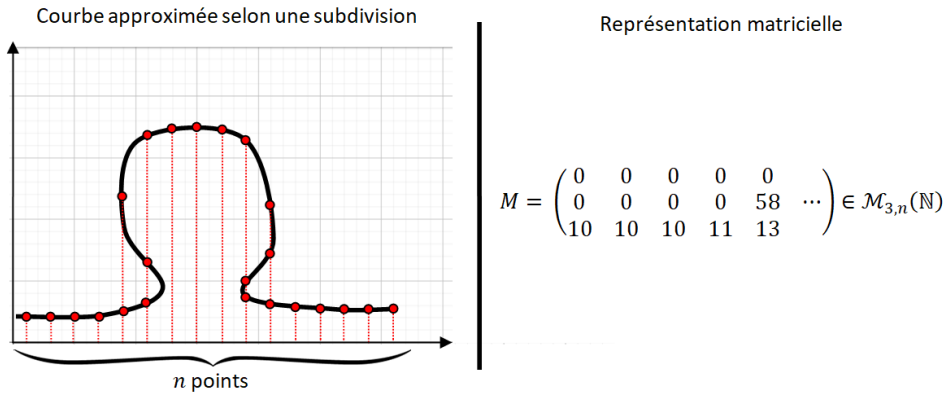

FIGURE 4 – Conversion en une instance de PUZZLE

3 Démonstration de NP-complétude

Dans cette section, nous démontrons la NP-complétude du problème PUZZLE ainsi que celles de plusieurs variantes plus restrictives introduites dans [7]. Les problèmes PUZZLE-ORIENTATION et PUZZLE-POSITION considèrent respectivement des instances pour lesquelles l'orientation et la position des pièces est fournie. Enfin, on introduit trois autres problèmes - CONTOUR, CONTOUR-ORIENTATION et CONTOUR-POSITION - variantes des problèmes précédents pour lesquels la question se réduit à l'existence d'un contour.

3.1 Problème CONTOUR

Théorème 3.1:
Le problème CONTOUR est NP-complet

Preuve :

— Le problème CONTOUR est dans NP. En effet, il suffit de remarquer qu'étant donné un certificat C qui correspond à la donnée d'un ordre et d'une orientation sur les $2m + 2n - 4$ pièces, il faut vérifier que :

- les numéros associés aux bords adjacents sont bien opposés l'un de l'autre : $O(m + n)$.
- les bords extérieurs sont bien numérotés par 0 : $O(m + n)$.
- le contour possède les bonnes dimensions : $O(m + n)$.

Ces vérifications s'effectuent bien en temps polynomial.

— Montrons que CONTOUR est NP-difficile en démontrant PARTITION $\leq_p$ CONTOUR. On se donne $E = \{x_1, ..., x_n\} \in \mathbb{N}^n$ un ensemble d'entiers naturels. On va effectuer la construction représentée en figure 5.

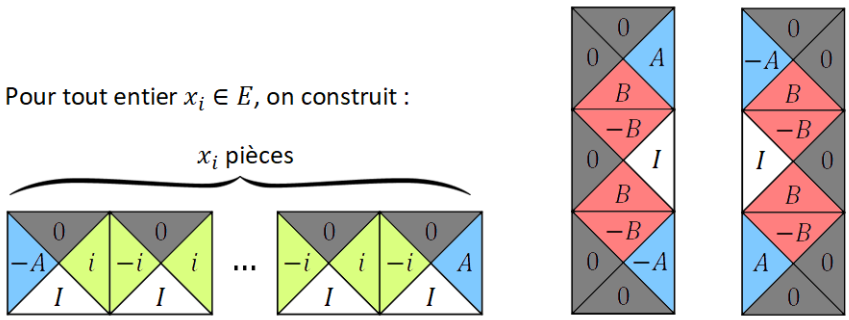

FIGURE 5 – Construction pour la réduction PARTITION $\leq_p$ CONTOUR

Pour chaque entier $x_i \in E$, on construit une ligne de x_i pièces qui ne peuvent pas être séparées dans la solution complète (si elle existe) en raison du nombre i qui est associé de manière unique à chaque x_i de E. On construit en plus de cela six pièces qui correspondront au bord droit et bord gauche. On prend pour instance de CONTOUR cet ensemble de pièce ainsi que le couple $\left(\frac{1}{2}\sum_{x_i \in E} x_i + 2, 3\right)$.

S'il existe un sous-ensemble I solution de PARTITION, alors les blocs de pièces correspondant aux entiers de I se juxtaposent dans un ordre quelconque pour former le bord supérieur du contour, les blocs correspondants aux entiers restant forment alors le bord inférieur, il existe ainsi un contour valide.

Réciproquement, s'il existe un contour valide, il suffit de prendre pour sous-ensemble I les tailles des blocs qui forme le bord supérieur du puzzle, il existe ainsi une partition solution. □

On trouve en figure 6 une construction complète pour une instance simple de PARTITION.

Instance de PARTITION admettant une solution :

$E = \{2,3,4,3,4\}$

$I_1 = \{2,3,3\}, I_2 = \{4,4\}$

Instance de CONTOUR correspondante :

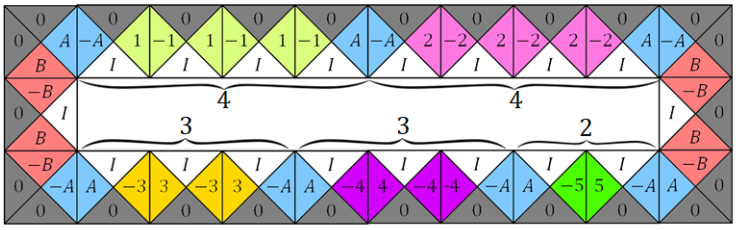

FIGURE 6 – Construction du théorème 3.1 sur un exemple

En modifiant légèrement la démonstration de la NP-complétude du problème CONTOUR, on pourra en déduire une démonstration du théorème 3.3.

3.2 Problème PUZZLE

Théorème 3.2:
Le problème PUZZLE est NP-complet

Preuve :

— Le problème PUZZLE est dans NP, pour montrer cela, il suffit de remarquer qu'étant donné un certificat C qui correspond à la donnée d'un ordre sur les $m \times n$ pièces ainsi qu'une orientation pour chaque pièce, il faut vérifier que :
- les numéros associés aux bords adjacents sont bien opposés l'un de l'autre : $O(mn)$
- les bords extérieurs sont bien numérotés par 0 : $O(m+n)$.
- il y a bien $m-2$ ou $n-2$ pièces entre chaque paire de coins du bord : $O(m+n)$

— Montrons que PUZZLE est NP-difficile en démontrant : CONTOUR $\leq_p$ PUZZLE. Considérons une instance de CONTOUR, c'est-à-dire un couple (m,n) et un ensemble de $2m+2n-4$ pièces. On effectue la construction détaillée en figure 7.

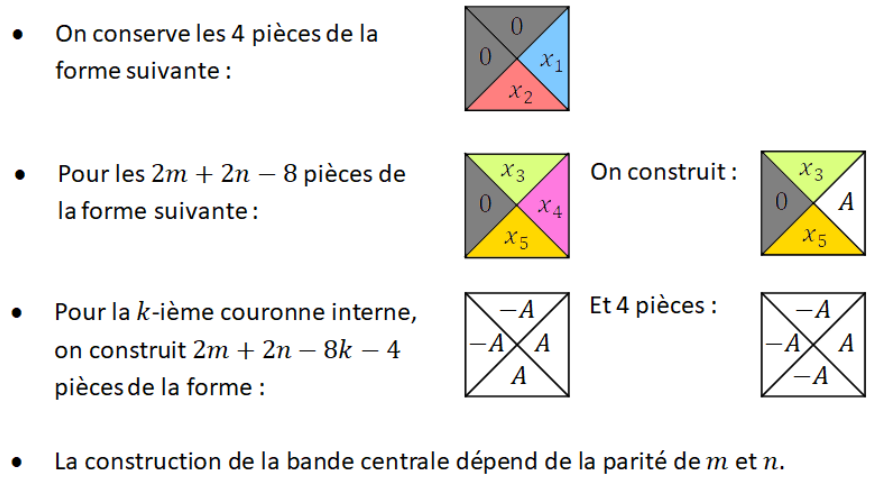

FIGURE 7 – Construction pour la réduction CONTOUR $\leq_p$ PUZZLE

On prend pour instance de PUZZLE cet ensemble de $m \times n$ pièce et le couple (m,n). En remplaçant le numéro du bord qui est dirigé vers l'intérieur, on ne retire aucune contrainte,

cette valeur n'influence pas l'existence ou non d'un contour. La construction assure que le carré central sera toujours constitué de pièces dont les bords sont numérotés uniquement par $\pm A$ (comme ces pièces ne contiennent pas de 0, elles sont nécessairement internes). Il est immédiat qu'on peut déduire une solution de PUZZLE à partir d'un contour valide. Réciproquement, si on dispose d'une solution de PUZZLE, il suffit de retirer les pièces centrales puis de remplacer A par l'indice originel de chaque pièce du contour. □

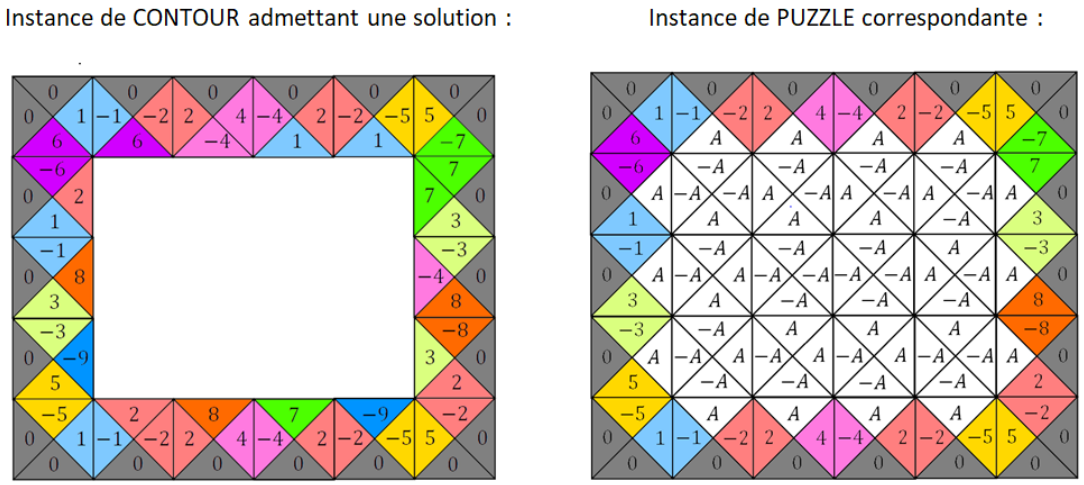

FIGURE 8 – Construction du théorème 3.2 sur un exemple

Une autre démonstration de la NP-complétude du problème PUZZLE a déjà été formulée par Takenaga et Walsh [3]. Leur méthode est cependant plus complexe car elle repose sur une réduction directe au problème 3-SAT.

3.3 Problème PUZZLE-ORIENTATION

Théorème 3.3:
Le problème PUZZLE-ORIENTATION est NP-complet

Preuve : La preuve est identique à celle du théorème 3.1, à l'exception que les entiers sont représentés par des pièces internes, et que leur orientation est fixée par les bords indicés par I et $-I$.

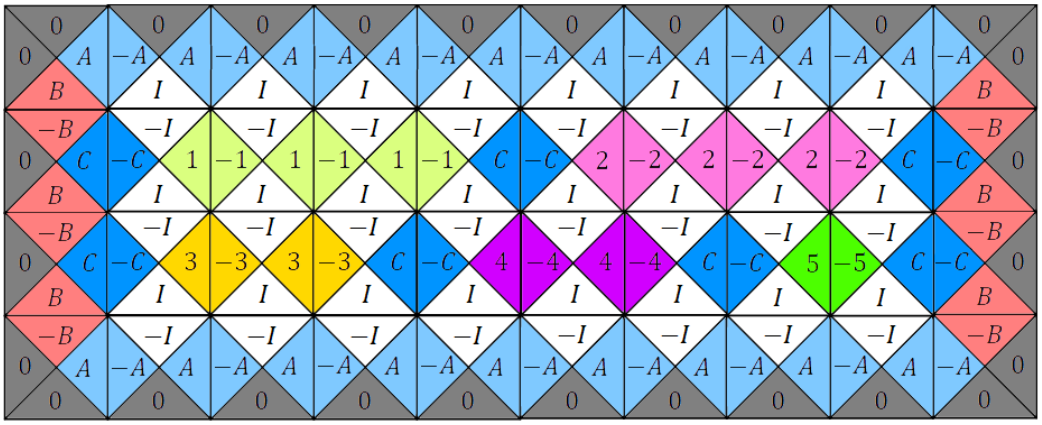

FIGURE 9 – Construction pour l'instance introduite en figure 6

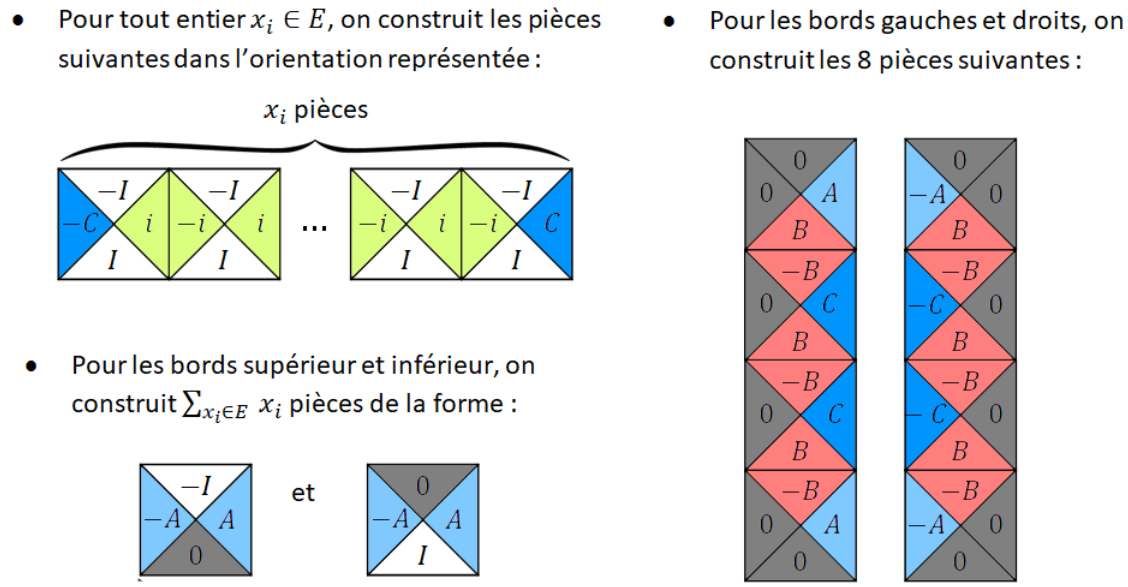

FIGURE 10 – Construction pour la réduction PARTITION $\leq_p$ PUZZLE-ORIENTATION

3.4 Problème PUZZLE-POSITION

Théorème 3.4:
Le problème PUZZLE-POSITION est NP-complet

Preuve :

— Le problème PUZZLE-POSITION est dans NP. (Mêmes vérifications que celles de la preuve du théorème 3.2)

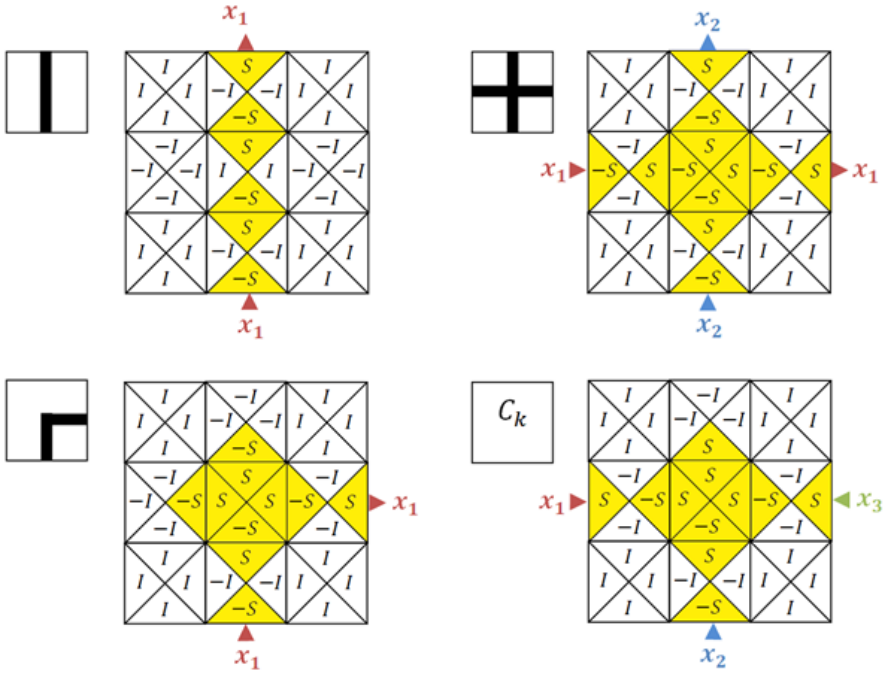

FIGURE 11 – Blocs 3×3 utilisés dans la construction

— Montrons que PUZZLE-POSITION est NP-difficile en démontrant : 1-in-3 SAT $\leq_p$ PUZZLE-POSITION.
Considérons une instance de 1-in-3 SAT, c'est-à-dire une formule logique sous forme normale conjonctive $\varphi = C_1 \wedge \cdots \wedge C_n$.
L'idée de la preuve est simple : pour chaque clause C_k, on utilise un motif de pièce, appelé "récepteur", qui ne peut être orienté correctement que si exactement l'une des entrées est vraie (voir figure 11).
Pour représenter les variables prenant des valeurs booléennes, on utilise un motif qui autorise uniquement deux orientations différentes de la pièce, qui correspondent à vrai ou faux. On construit alors, pour chaque occurrence d'une même variable, un bloc de pièces 3×3 comme ceux de la figure 12, qu'on appellera "émetteurs" dans la suite. (Il est important de remarquer qu'en fonction de l'emplacement du bloc 3×3 dans la construction finale, les signes devant l'indice I changent.)

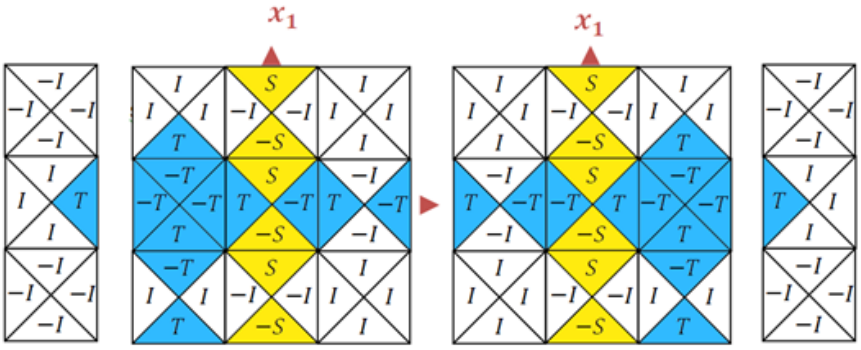

FIGURE 12 – Construction pour une variable : "émeteurs"

La pièce centrale de chaque bloc émetteur peut prendre deux formes : $(T, S, -T, -S)$ qui correspond à une occurrence positive, et $(T, -S, -T, S)$ qui correspond à une occurrence négative de la variable. Les pièces contenant l'entier T assurent que toutes les occurrences de la variable aient la même valeur de vérité.

Enfin, il ne reste qu'à relier les émetteurs aux récepteurs. On effectue pour cela la construction de la figure 13.

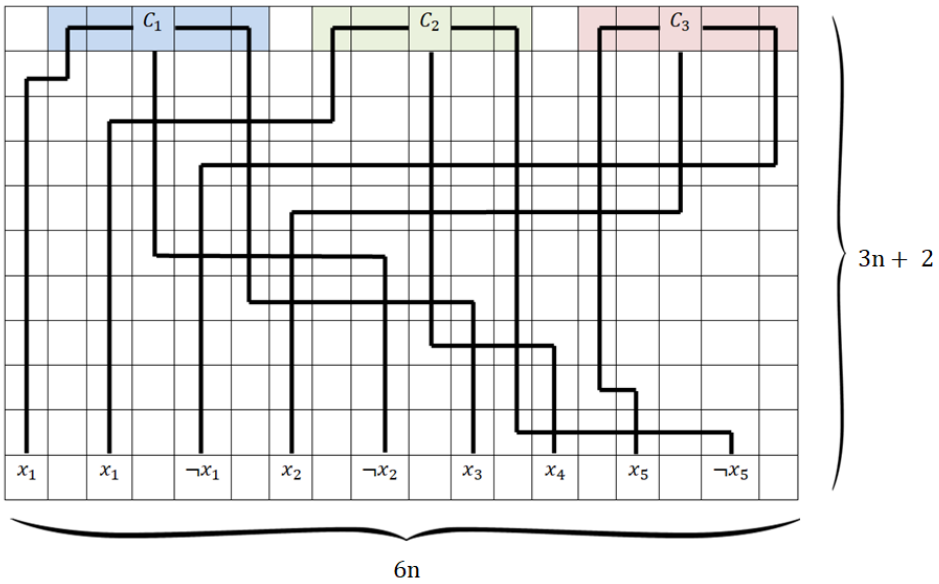

FIGURE 13 – Construction pour $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_4 \vee \neg x_5) \wedge (x_5 \vee x_2 \vee \neg x_1)$, $n = 3$

Cette construction garantit l'absence de chevauchement entre les pièces qui transmettent les différentes valeurs de vérité. Pour le k-ième émetteur (dans l'ordre d'apparition de la construction), on transmet la valeur de vérité verticalement jusqu'à la ligne $k+1$, puis on la transmet horizontalement jusqu'à la colonne correspond à la clause dans laquelle la variable apparait. Les émetteurs sont placés sur les colonnes d'indice pair, les récepteurs sur les colonnes d'indice impair, de plus les transmissions horizontales se font sur des lignes distinctes, évitant ainsi les chevauchements.

Cependant, la construction actuelle n'est pas exactement une instance de PUZZLE-POSITION car la dernière ligne contient des pièces dont l'indice du bord inférieur peut varier. Pour clore la figure, une solution consiste à ajouter à la construction une partie inférieure qui correspond à sa symétrie par rapport à la dernière ligne. Les valeurs de vérité transmises dans la partie inférieure sont alors l'opposé des valeurs de vérité réelles. Il reste enfin à modifier les récepteurs de la partie inférieure pour qu'ils admettent une configuration valide si et seulement si exactement deux entrées sont vraies. Les pièces du contour actuel ont alors leur valeur extérieure fixe $\pm I$ (ou $\pm T$ pour les quatre pièces en contact avec un émetteur), il suffit donc de rajouter un contour avec des pièces comportant des 0 pour valeur extérieure. On a ainsi créé une instance de PUZZLE-POSITION de taille $18n(9n+6)$, où n est le nombre de variables de φ.

Supposons qu'il existe une valuation σ rendant φ vraie telle que chaque clause contiennent exactement une variable évaluée à vraie. Alors, il suffit, pour chaque récepteur, d'orienter la face indicé par $-S$ vers la face correspondant à l'unique variable évaluée à vraie. Cela impose l'orientation de toutes les pièces de la construction, et comme une même variable possède la même valuation dans toutes les clauses dans lesquelles elle apparait, il n'y a pas de conflit entre les pièces constituant les émetteurs.
Réciproquement, si la construction admet une configuration valide, on peut construire une valuation valide pour l'instance de 1-in-3 SAT correspondante en associant à chaque variable la valeur de vérité prise par l'émetteur lui correspondant. □

Ces résultats nous indiquent que même des variantes du problème PUZZLE imposant bien plus de contraintes que le problème initial demeurent NP-complètes. Il semble donc difficile d'établir un algorithme efficace pour PUZZLE. Dans les sections suivantes, nous allons étudier les variantes PUZZLE-ORIENTATION et PUZZLE-POSITION pour tenter d'établir un algorithme exact pour résoudre efficacement les instances de PUZZLE.

4 Problème PUZZLE-ORIENTATION

4.1 α–approximation

L'objectif de cette section est d'étudier le cas où nous avons connaissance de l'orientation des pièces. Nous supposerons que toutes les instances considérées dans cette partie admettent au moins une solution. Cette hypothèse permettra une plus grande souplesse sur les démonstrations de cette section en modifiant légèrement la définition classique d'α-approximation. De plus, cette hypothèse est raisonnable, car elle correspond au cas pratique pour lequel le puzzle admet toujours une solution.

Définition (pièce valide) : Une pièce est dite valide si elle est disposée dans une configuration partielle telle qu'elle ne possède aucune pièces voisines, ou si ses bords sont compatibles avec ceux des pièces voisines. Les pièces placées sur le contour doivent en plus respecter la contrainte d'une pièce voisine virtuelle portant la valeur 0.

Définition (α-approximation) : Soit $\alpha : (N^*)^2 \longrightarrow [0,1]$. On dit qu'un algorithme est une $\alpha(m,n)$-approximation de PUZZLE-ORIENTATION lorsque pour toute instance (admettant une solution) de taille $(m,n) \in (N^*)^2$, l'algorithme permet de placer au minimum $\alpha(m,n) \times mn$ pièces valides en temps polynomial.

Proposition 4.1:
Le problème PUZZLE-ORIENTATION admet une $\frac{1}{2}$-approximation.

Preuve : Ce résultat découle de la constatation suivante : en plaçant la moitié des pièces dans une configuration en damier (ce qui se fait en temps linéaire en le nombre de pièces), aucune contrainte n'est à respecter car les pièces n'ont aucune voisine. On fera attention à prendre les pièces placées sur le contour parmi celle comportant un 0.
Notons $N_{m,n}$ le nombre de pièces ainsi placées, alors :

$$N_{m,n} = \left\lfloor \frac{m}{2} \right\rfloor \left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{m}{2} \right\rceil \left\lceil \frac{n}{2} \right\rceil$$

Or, on a la relation suivante pour tout $n \in \mathbb{Z}$:

$$n = \left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor$$

Ainsi :

$$N_{m,n} = \left\lfloor \frac{m}{2} \right\rfloor \left\lfloor \frac{n}{2} \right\rfloor + \left(m - \left\lfloor \frac{m}{2} \right\rfloor\right) \left(n - \left\lfloor \frac{n}{2} \right\rfloor\right)$$

On peut alors minorer cette quantité avec la relation :

$$\forall x \in \mathbb{R} : x - 1 < \lfloor x \rfloor \leq x$$

Ce qui donne :

$$N_{m,n} \geq mn - m\frac{n}{2} - n\frac{m}{2} + 2\frac{m}{2}\frac{n}{2} = \frac{mn}{2}$$

Donc, on dispose bien d'une $\frac{1}{2}$-approximation. □

Théorème 4.2:
Le problème CONTOUR-ORIENTATION est dans P.

Preuve : Ayant connaissance de l'orientation des pièces, on peut alors déterminer à quel bord elles appartiennent en regardant la position de l'indice 0 sur la pièce. La position des coins dans le puzzle est quant à elle entièrement déterminée par leur orientation, par exemple, le coin contenant un 0 sur sa face supérieure et sa face gauche ne peut être placé qu'en haut à gauche dans le puzzle.

Pour déterminer si les pièces constituant chacun des bords admettent une organisation valide, nous allons effectuer la construction suivante. Considérons l'ensemble formé par les n pièces d'un bord, par exemple le bord supérieur (la démonstration est analogue dans les autres cas) :

$$P_{sup} = \{(i_g, i_d) \mid i \in [\![1, n]\!]\}$$

Avec i_g (resp. i_d) l'indice porté par le bord gauche (resp. droit) de la pièce i. On considère b_d l'indice de la face gauche du coin supérieur droit et b_g l'indice de la face droite du coin supérieur gauche. On construit le graphe orienté $G = (S, A)$ avec :

— $S = \{s_i, s_j, e_i, e_j \mid (i, j) \in P_{sup}\} \cup \{s_{b_d}, s_{b_g}, e_{b_d}, e_{b_g}\}$

— $A = \{(s_i, e_j) \mid (i, j) \in P_{sup}\} \cup \{(e_i, s_{-i}) \mid ((a, i), (-i, b)) \in (P_{sup})^2\} \cup \{(s_{b_d}, e_{b_g})\}$

Autrement dit, on crée quatre sommets pour chaque indice (en valeur absolue) apparaissant sur les pièces du bord concerné. Les arêtes de la forme (s_i, e_j) représentent les pièces du bord, les arêtes de la forme (e_i, s_{-i}) représentent les liens entre ces pièces. Enfin, on rajoute une dernière arête qui représente un lien entre le coin droit et le coin gauche. Il est important de noter que l'ensemble S est construit sans doublons, tandis que l'ensemble A autorise les doublons. Cela est dû au fait que deux pièces peuvent posséder le même indice sur une même face.
Une représentation de cette construction dans un cas simple est disponible en figure 14.

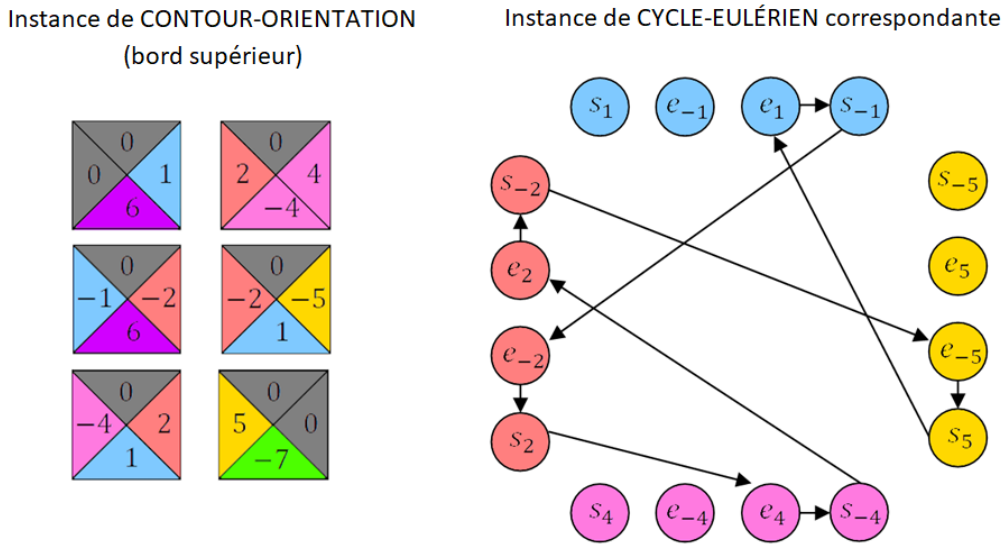

FIGURE 14 – Réduction CONTOUR-ORIENTATION $\leq_p$ CYCLE-EULÉRIEN

S'il existe un contour valide, que l'on note :

$$b_g \longleftrightarrow (1_g, 1_d) \longleftrightarrow \cdots \longleftrightarrow (n_g, n_d) \longleftrightarrow b_d$$

Alors, par construction, on aura le chemin eulérien suivant dans le graphe G, qui contient bien toutes les arêtes construites :

$$e_{b_g} \longrightarrow s_{1_g} \longrightarrow e_{1_d} \longrightarrow s_{2_g} \longrightarrow \cdots \longrightarrow s_{n_g} \longrightarrow e_{n_d} \longrightarrow s_{b_d} \longrightarrow e_{b_g}$$

Réciproquement, si le graphe G admet un cycle eulérien, alors il est nécessairement de la forme :

$$e_{b_g} \longrightarrow s_{1_g} \longrightarrow e_{1_d} \longrightarrow s_{2_g} \longrightarrow \cdots \longrightarrow s_{n_g} \longrightarrow e_{n_d} \longrightarrow s_{b_d} \longrightarrow e_{b_g}$$

En effet, par construction, les seules arêtes sortantes des sommets de la forme s_i sont entrantes pour des sommets de la forme e_j (avec $i \neq j$). De même pour des arêtes sortantes des sommets de la forme e_i, qui sont entrantes pour des sommets de la forme e_{-i}. Cela permet de garantir l'alternance entre les arêtes correspondant à une pièce, et les arêtes correspondant à un lien entre

deux pièces. Il existe donc un contour.

Ainsi CONTOUR-ORIENTATION $\leq_p$ CYCLE-EULÉRIEN, or CYCLE-EULÉRIEN $\in P$, donc CONTOUR-ORIENTATION $\in P$ □

Corollaire 4.3:
Le problème PUZZLE-ORIENTATION admet une $\left(\frac{1}{2}+\frac{1}{n}+\frac{1}{m}-\frac{6}{mn}\right)$-approximation.

Preuve : En s'inspirant de la démonstration du théorème 4.2, il est possible d'établir une adaptation de l'algorithme de Hierholzer permettant de mettre en évidence un cycle eulérien en temps linéaire en la taille du graphe.
Notre version de l'algorithme permet d'établir une configuration valide des contours en temps polynomial : $O(m+n)$, soit $2m+2n-4$ pièces. Il ne reste qu'à compléter le carré central avec la construction en damier précédente : $O(mn)$. ;
Il faut tout de même remarquer qu'il n'existe pas nécessairement de pièce compatible avec les coins du puzzle privé de son contour, car une pièce placée dans ces emplacements doit vérifier deux contraintes. On retirera donc au maximum 4 pièces. Pour les emplacements en contact uniquement avec une pièce du contour, il n'y a qu'une contrainte à respecter, or, l'existence d'une solution garantit l'existence de pièces vérifiant ces contraintes.
Notons $M_{m,n}$ le nombre de pièces ainsi placées :

$$M_{m,n} \geq 2(m-1)+2(n-1)+N_{m-2,n-2}-4$$

$$\geq \frac{nm+2(m+n)-12}{2}$$

Ceci permet d'obtenir une $\alpha(m,n)$-approximation, avec :

$$\alpha(m,n)=\frac{nm+2(m+n)-12}{2mn} \quad \square$$

Théorème 4.4:
Le problème PUZZLE-ORIENTATION admet une $\left(\frac{5}{9}-\frac{2(m+n)}{9}\right)$-approximation.

Preuve : Dans la démonstration suivante, rien ne garantit qu'on puisse trouver un contour compatible avec la disposition des pièces intérieure, on ne va donc pas pouvoir placer la totalité du contour. Cela ne change pas le comportement asymptotique de $\frac{\alpha(m,n)}{mn}$.
On va placer des blocs de 2×2 pièces comme dans la configuration de la figure 15, en distinguant neuf cas selon la congruence de m et n modulo 3.

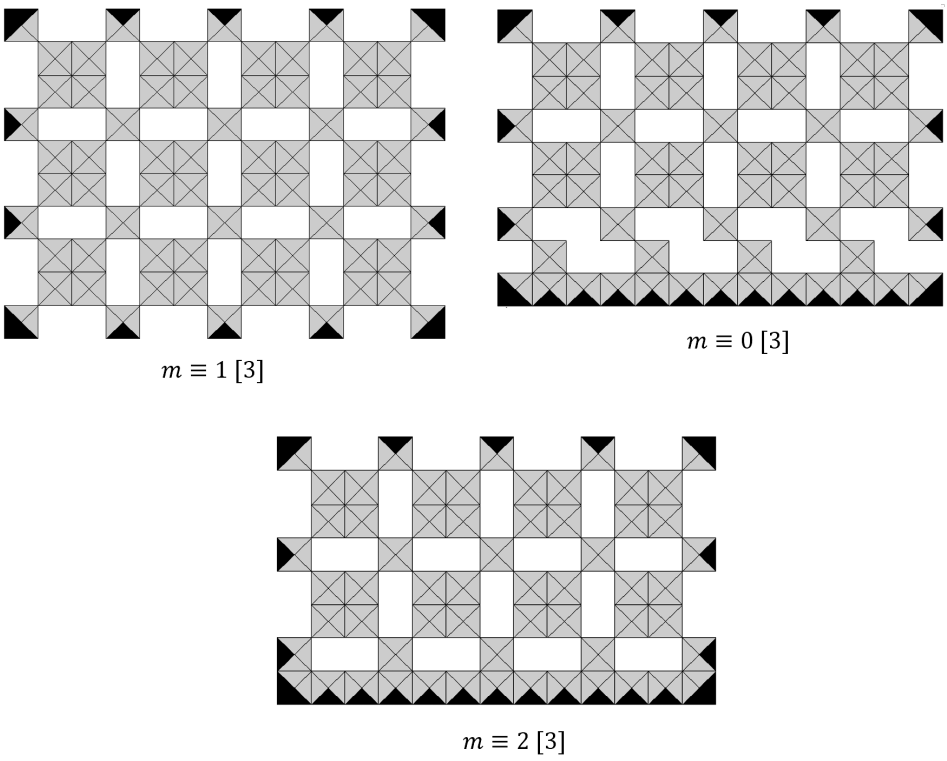

FIGURE 15 – Configurations possibles pour $n \equiv 1[3]$

On génère dans un premier temps tous les sous-ensembles de 4 pièces correspondant à une configuration 2×2 valide. On peut faire cela en $O((mn)^3)$ à l'aide de l'algorithme en annexe 8.3. On note $E_{2,2}$ cet ensemble. Comme une solution existe, on peut minorer le nombre de sous-ensembles de 4 pièces compatibles disjoints :

$$Card(SetPacking(E_{2,2})) \geq \left\lfloor \frac{m}{2} \right\rfloor \left\lfloor \frac{n}{2} \right\rfloor$$

En notant $N_{2,2}$ la quantité de sous-ensembles de taille 2×2 nécessaires pour la construction, on a :

$$N_{2,2} = \left\lfloor \frac{m-1}{3} \right\rfloor \left\lfloor \frac{n-1}{3} \right\rfloor$$

Or, nous disposons d'une $\frac{2}{k}$-approximation (au sens classique) de k-Set packing, une version de Set packing ou les sous-ensembles considérés sont de taille au plus k. On cherche, dans notre cas, le plus grand ensemble de sous-ensemble de $E_{2,2}$ deux à deux disjoints, or l'algorithme dont le pseudo code est fourni en annexe 8.3 nous donne un tel ensemble de cardinal vérifiant :

$$\forall (m,n) \in (\mathbb{N}^*)^2, \frac{1}{2} Card(SetPacking(E_{2,2})) \geq \frac{1}{2} \left\lfloor \frac{m}{2} \right\rfloor \left\lfloor \frac{n}{2} \right\rfloor \geq N_{2,2}$$

On obtient donc suffisamment de blocs de pièces pour réaliser la construction de la figure 15.

Des neuf configurations possibles, celle qui permet de placer le moins de pièce est $m \equiv n \equiv 1[3]$. Dans ce cas, on a placé exactement :

$$N = 5 \left\lceil \frac{n-1}{3} \right\rceil \left\lceil \frac{m-1}{3} \right\rceil + \left\lceil \frac{n-1}{3} \right\rceil + \left\lceil \frac{m-1}{3} \right\rceil \geq \frac{5(m-1)(n-1)}{9} + \frac{m+n+1}{3}$$

$$\geq \frac{5mn - 2(m+n)}{9mn}$$

Ceci permet donc d'obtenir une $\alpha(m,n)$-approximation, avec :

$$\alpha(m,n) = \frac{5}{9} - \frac{2(m+n)}{9} \quad \square$$

Remarque : Il n'est pas possible de faire mieux en rassemblant les pièces par paquets de 3×3 par exemple. En effet, en notant $k \times k'$ la dimension de ces sous-ensembles, il faut pouvoir vérifier l'inégalité :

$$\frac{2}{kk'} Card(SetPacking(E_{k,k'})) \geq N_{k,k'} \Longrightarrow \frac{2}{kk'} \left\lfloor \frac{m}{k} \right\rfloor \left\lfloor \frac{n}{k'} \right\rfloor \geq \left\lfloor \frac{m-1}{k+1} \right\rfloor \left\lfloor \frac{n-1}{k'+1} \right\rfloor$$

pour que notre $\frac{2}{k}$-approximation de k-Set Packing trouve suffisamment de sous-ensembles disjoints. Or pour $k = 2$ et $k' = 3$, l'inégalité n'est vérifiée que pour certaines valeurs de m et n, et pour tout autre couple $k \leq 3$, $k' \leq 3$, cette inégalité n'est plus vérifiée.

4.2 Algorithme exact

On peut utiliser un backtracking pour résoudre une instance de cette variante de PUZZLE, ce qui est plutôt efficace pour des instances ayant, pour tout i, j, peu de répétitions de pièces de la forme $(i, j, \_, \_)$. En particulier, un backtracking remplissant ligne par ligne un puzzle devient linéaire s'il n'y a pas deux occurrences d'une telle pièce.

Lorsque ce n'est pas le cas, un autre algorithme efficace en pratique consiste à construire $E_{2,2}$, définit dans la preuve du théorème 4.4 (dans la suite, une "k-pièce" fera référence à un bloc de pièces de taille $k \times k$). Cette opération permet de diminuer le problème à une taille $\frac{mn}{4}$, même s'il y a plus de 2-pièces que nécessaire, et qu'elles ne peuvent pas nécessairement apparaitre en même temps dans une solution.

Chaque 2-pièce possède 4 faces indicées par les couples des indices qui les composent. On peut ainsi remarquer que si la 2-pièce p contient pour indice le couple (i, j), il faut que $(-i, -j)$ apparaisse dans une autre 2-pièce ne partageant aucune sous-pièce avec p, sinon on peut éliminer p. Ce résultat permet de diminuer le cardinal de $E_{2,2}$.

On peut alors répéter le procédé et déterminer $E_{4,4}$, en faisant attention à générer uniquement des 4-pièces contenant à la fois des 2-pièces toutes distinctes et des pièces toutes distinctes... Et le procédé continue ainsi jusqu'à obtenir toutes les $2^{\lfloor \log_2(\min(m,n)) \rfloor}$-pièces possibles, il en existe nécessairement 4 qui s'intersectent pour former le puzzle complet.

Cependant, une autre optimisation est possible, en effet, le nombre de conditions que doivent vérifier les k-pièces devient rapidement très grands, et en pratique, pour des valeurs de $k > 2$, on a $Card(E_{2^k,2^k}) \approx \frac{Card(E_{2^{k+1},2^{k+1}})}{4}$, c'est-à-dire que le nombre de pièces qu'on peut élaguer diminue rapidement, il n'est donc plus intéressant de faire toutes les vérifications détaillées précédemment. Dans la plupart des cas pratiques, il sera plus judicieux de lancer un backtracking sur les éléments de $E_{4,4}$.

5 Problème PUZZLE-POSITION

5.1 α-approximation

Définition (α-approximation) : Soit $\alpha : (N^*)^2 \longrightarrow [0,1]$. On dit qu'un algorithme est une $\alpha(m,n)$-approximation de PUZZLE-POSITION lorsque pour toute instance (admettant une solution) de taille $(m,n) \in (N^*)^2$, l'algorithme fait correspondre $\alpha(m,n) \times (2mn + m + n)$ paires de faces voisines en temps polynomial.

Proposition 5.1:
Le problème CONTOUR-POSITION est dans P.

Preuve : Ce résultat est évident, la connaissance de la position des pièces du contour donne la connaissance de leur orientation grâce aux faces indicées par 0 qui doivent être orientées vers l'extérieur.

Théorème 5.2:
Le problème PUZZLE-POSITION admet une $\left(\dfrac{4n + 4m + nm - 9}{2mn + m + n}\right)$-approximation

D'après la proposition 2.1, il est possible d'établir les contours en temps polynomial. On peut alors effectuer le parcours de la figure 15 et former un graphe orienté de la manière suivante :

— Un sommet correspond à une orientation possible d'une pièce (respectant les contraintes du contour, si la pièce en touche un).

— L'arête orientée (i,j) représente le fait que la face de sortie (dans le sens du parcours) de i et la face d'entrée de j sont compatibles.

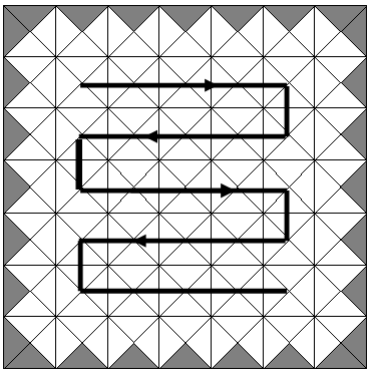

FIGURE 16 – Parcours des pièces internes

Une fois le graphe créé, on ajoute des arêtes virtuelles entre toutes les orientations possibles de la première pièce et toutes celles de la dernière (dans le sens de parcours). On peut alors trouver en cycle eulérien dans ce graphe, qui existe nécessairement, car une solution existe par hypothèse. En orientant les faces selon l'orientation correspondante, on s'assure que toutes les paires de faces empruntées par le parcours se correspondront. Le fait que le graphe soit orienté permet de garantir qu'une seule orientation sera choisie pour chaque pièce. On détermine ainsi une $\alpha(m,n)$ approximation, avec :

$$\alpha(m,n) = \frac{4n + 4m + nm - 9}{2mn + m + n} \quad \square$$

Remarque : Le parcours de la figure 15 est arbitraire. Quelle que soit la forme du parcours qu'on effectue, le nombre de contraintes reste le même.

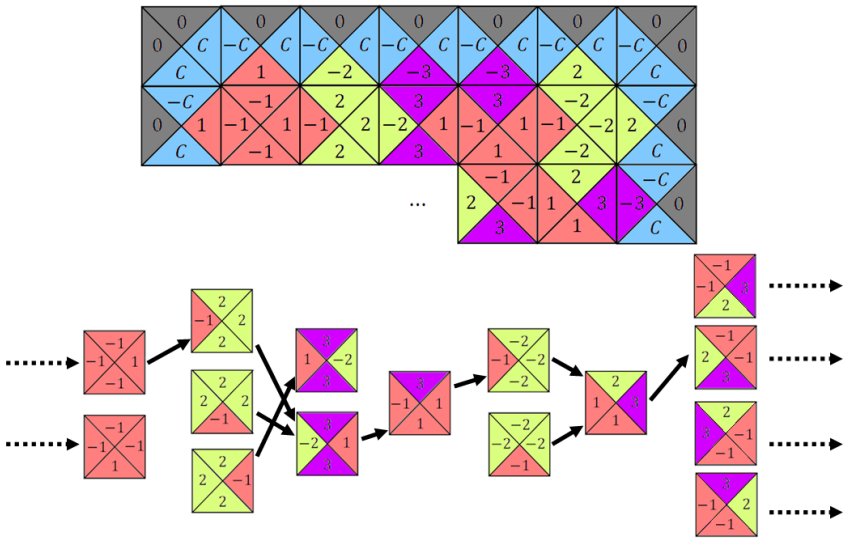

FIGURE 17 – Représentation partielle de la construction du théorème 5.2

5.2 Algorithme efficace dans les cas pratiques

5.2.1 Fonctionnement de l'algorithme :

On dispose, d'après la proposition 5.1 d'un algorithme en $O(m+n)$ qui permet de trouver l'orientation des pièces constituant le contour du puzzle (qu'on note $OrienterContour$). On sait aussi que les pièces invariantes par rotation, de la forme (i,i,i,i), sont nécessairement correctement orientées.
On effectue alors un parcours des pièces restantes pour vérifier si les orientations connues imposent leur orientation. Tant que ce parcours découvre au moins une pièce dont l'orientation est imposée, on effectue un nouveau parcours.
Si une pièce n'a qu'une orientation possible vérifiant les conditions imposées, alors on la place dans cette orientation (avec $OrienterPiece$) et on l'ajoute à la liste des pièces dont l'orientation est connue.

Algorithme 1 : Propagation des contraintes

```
OrienterContour(contour);
OrientationsConnues ← [contour];
changement ← vrai;
pour piece ∈ puzzle\contour faire
    si piece = (i, i, i, i) alors
        OrientationsConnues ← OrientationsConnues ∪{piece};
    fin
fin
tant que changement faire
    changement ← faux ;
    pour piece ∈ puzzle\OrientationsConnues faire
        NbOrientation ← 0 ;
        pour i = 0 à 3 faire
            si OrientationCorrecte(piece, OrientationsConnues) alors
                NbOrientation ← NbOrientation + 1;
                Tourner(piece);
            fin
        fin
        si NbOrientation = 1  ou  (NbOrientation = 2  et piece = (i, j, i, j)) alors
            OrienterPiece(piece, OrientationsConnues);
            OrientationsConnues ← OrientationsConnues ∪{piece};
            changement ← vrai ;
        fin
    fin
fin
```

5.2.2 Terminaison et complexité

Chaque itération de la boucle **tant que** fait décroitre la quantité entière $Card(puzzle\backslash OrientationsConnues)$. Cette quantité finit donc soit par être constante, l'algorithme s'arrête car aucune pièce de cet ensemble n'a d'orientation imposée, soit par valoir 0, l'algorithme termine et toutes les pièces sont correctement orientées.

L'Algorithme 1 à une complexité temporelle en $O((mn)^2)$. En effet :

- On génère la liste des pièces dont l'orientation est connue : $O(mn)$.
- D'après le raisonnement précédent, on effectue au plus $(m-2)(n-2)$ itérations de la boucle **tant que**. Au sein de cette boucle, on parcourt toutes les pièces restantes.

La complexité est donc bien en $O(mn + (mn)^2) = O((mn)^2)$.

5.2.3 Instances pour lesquelles l'algorithme ne fournit pas de solution

En théorie, cet algorithme ne permet pas nécessairement de trouver une solution. On peut le remarquer en considérant le puzzle dont les pièces internes sont toutes de la forme (i, i, i, j) ou $(-i, -i, -i, -j)$.

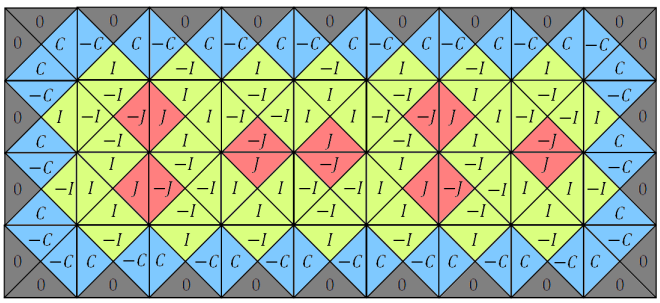

FIGURE 18 – Instance pour laquelle l'algorithme échoue.

Remarque : Cette construction fournit aussi un exemple d'instance de PUZZLE-POSITION possédant un nombre exponentielle de solutions en la taille du puzzle. Pour la construction précédente de taille $4 \times n$, il y a exactement F_{n-2} solutions, où $(F_n)_{n\in\mathbb{N}}$ est la suite de Fibonacci.

5.2.4 Analyse dans le cas pratique :

Théorème :
Les instances de PUZZLE-POSITION contenant strictement moins de 4 pièces de la forme (i, i, i, j) avec $i \neq j$, sont résolue en $O((mn)^2)$ par l'algorithme 1.

Preuve : Procédons par induction pour démontrer que c'est bien le cas lorsqu'il n'y a pas de telles pièces. Admettons que l'algorithme suit le parcours introduit en figure 16. Si la pièce actuelle est la pièce du début de parcours, alors, en notant i, j les indices des faces contraintes par les pièces du contour, on doit étudier deux cas :

— Si $i \neq j$, alors, pour avoir plusieurs orientations possibles, il faut avoir une pièce de la forme $(-i, -j, -i, -j)$, or cette pièce est invariante par double rotation, les deux orientations correspondent donc à la même disposition, qui est donc déterminée de manière unique.

— Si $i = j$, alors, pour avoir plusieurs orientations possibles, on a soit une pièce de la forme $(-i, -i, -i, -i)$, qui est invariante par rotation, donc correspond à la même disposition pour chaque orientation, soit $(-i, -i, -i, -k)$ avec $k \neq j$, mais par hypothèse, de telles pièces n'appartiennent pas au puzzle.

La première pièce est déterminée de manière unique. Si maintenant, la pièce actuelle est quelconque, comme toutes les pièces précédentes dans le parcours sont orientées de manière certaine, la pièce actuelle subie au minimum deux contraintes imposées par sa voisine de gauche et sa voisine de droite qui sont déjà placées. Par le même raisonnement que précédemment, elle est orientée de manière unique.

Supposons maintenant qu'on place uniquement 3 pièces de la forme (i, i, i, j) avec $i \neq j$. D'après l'induction précédente, ont fini par imposer deux contraintes de face à la première pièce de la forme (i, i, i, j) apparaissant dans le parcours. Supposons dans un premier temps que cette pièce soit la seule de cette forme sur sa ligne, l'algorithme orientera de manière unique les pièces restantes de cette ligne, en commençant dans le pire cas par celle en contact avec le contour. Cela finira par imposer une troisième contrainte sur la pièce considérée en fixant sa voisine de droite, ce qui suffit à l'orienter totalement. Si, à l'inverse, la pièce n'est pas la seule de cette forme sur la ligne, on peut effectuer le même raisonnement sur la colonne. Enfin, si la colonne et la ligne possède une voisine de la même forme, ça ne peut pas être le cas pour ces pièces, elles auront donc une orientation imposée, et le parcours finira par imposer l'orientation de cette dernière pièce. □

Cette démonstration permet de mieux comprendre l'efficacité de l'algorithme 1 en pratique, lorsqu'il y a plus de 2 couleurs de faces distinctes. Le théorème précédent ne fournit en plus qu'une condition nécessaire, on pourrait aussi prendre en compte le fait que deux possibilités existe pour l'orientation d'une pièce (i, i, i, j), seulement si cette pièce possède au moins deux voisines ayant elle-même au moins une occurrence de i et de j.

6 Algorithme exact pour le problème PUZZLE

L'approche classique pour résoudre ce problème est d'effectuer un backtracking. On choisit à chaque étape une des pièces restantes à placer parmi les emplacements disponibles ainsi que son orientation, si c'est impossible, on change un des choix précédents. On poursuit ce procédé jusqu'à trouver une solution.
Cependant, cette méthode devient vite inefficace pour des entrées de grande taille. En effet, pour une entrée de taille $m \times n$, il y a $(mn)! \times 4^{mn}$ positions possibles. De plus, l'instance "Eternity II" d'un problème similaire à PUZZLE est un exemple d'instance ayant beaucoup de configurations possibles - $(256)! \times 4^{256}$ - et n'ayant que très peu de solutions — environ 20,000 selon son inventeur [4,5].

6.1 Backtracking sur les positions

À l'aide de l'algorithme de propagation des contraintes établi dans la section précédente, nous pouvons déterminer une autre méthode pour résoudre des instances de PUZZLE possédant un grand nombre de couleurs (complexité quadratique en pratique).
Plutôt que d'effectuer un backtracking sur un arbre possédant potentiellement $(mn)!\times 4^{mn}$ branches, on va dans un premier temps effectuer un backtracking qui va placer les pièces dans une configuration particulière devant vérifier certaines conditions nécessaires pour que la configuration soit valide. Au moment du placement d'une pièce p, on va s'assurer que chaque voisine possède au minimum une couleur de p. De plus, si l'ajout de p permet à une pièce d'obtenir toutes ses voisines, on doit s'assurer que chaque couleur de p apparait au moins une fois dans les couleurs des voisines.

En procédant ainsi, on place les pièces sous la forme d'une instance de PUZZLE-POSITION, on peut alors appliquer l'algorithme de propagation des contraintes pour déterminer en temps $0((mn)^2)$ si cette instance admet une solution (il faudra faire attention à modifier l'algorithme 1 pour arrêter la recherche si $NbOrientation = 0$.) On parcourt ainsi un arbre de taille $(mn)!$, et pour les branches valides, on effectue, pour la plupart des instances, en moyenne $O((mn)^2)$ opérations, ce qui constitue une nette amélioration par rapport au backtracking naïf.

6.2 Conclusion

Bien que la méthode présentée dans ce document soit plus efficace en pratique qu'un backtracking naïf, l'algorithme ayant obtenu à ce jour les meilleurs résultats pour des problèmes similaires, tel qu'Eternity II, reste une forme de backtracking "amélioré". Le défaut majeur de notre méthode est qu'elle parcourt un premier arbre dans lequel la plupart des branches aboutissent à une disposition valide. Les contraintes ont dû être relâchées à chaque placement par rapport au backtracking original. La méthode utilisée pour améliorer l'élagage des branches de l'arbre consiste à maintenir en mémoire la quantité de pièces contenant le couple de faces (i, j), puis à choisir un parcours qui place les pièces dans les "coins" du plateau, voir figure 19, ce qui maximise les contraintes à respecter sur les occurrences des couples de faces, et permet d'élaguer l'arbre plus rapidement.

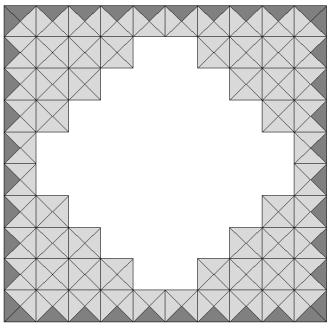

FIGURE 19 – Remplissage en "coin"

7 Annexes

7.1 Bibliographie

[1] FREEMAN H. ET GARDER L.,Jigsaw Puzzles : The Computer Solution of a Problem in Pattern Recognition, doi : 10.1109 / PGEC.1964.263781

[2] ANDERS MARTINSSON, Shotgun edge assembly of random jigsaw puzzles, https://arxiv.org/abs/1605.07151v2

[3] YASUHIKO TAKENAGA ET TOBY WALSH, Tetravex is NP-complete, https://arxiv.org/abs/0903.1147

[4] TIMO JOLIVET, Eternity II : puzzles, algorithmique et complexité, https://efreidoc.fr/L2%20-%20PL2/SDD/Projets/2007-08%20%3A%20Eternity%20II/Groupe%203/2007-08.projet.eternityII.rapport03.sdd.pdf

[5] LUDOVIC PATEY : Eternity II et variantes : https://ludovicpatey.com/media/research/patey-e2-rapport.pdf

[6] MICHAEL BRAND : No easy puzzles : Hardness results for jigsaw puzzles : https://doi.org/10.1016/j.tcs.2015.02.030

[7] VAHAN HUROYAN, GILAD LERMAN ET HAU-TIENG WU : Solving Jigsaw Puzzles bythe Graph Connection Laplacian : https://arxiv.org/pdf/1811.03188.pdf

7.2 Définition des problèmes

Dans tous les problèmes considérés dans la suite, la valeur 0 est réservée pour les pièces du contour du puzzle. On définit les problèmes suivants :

— PARTITION :

Instance : un ensemble fini d'entiers naturels $E = \{x_1, ..., x_n\}$.

Question : Peut-on partitionner E en deux ensembles de même somme, c'est-à-dire existe-t-il $I \subset E$ tel que $\sum_{x_i \in I} x_i = \sum_{x_i \notin I} x_i$?

— PUZZLE :

Instance : Un couple (m, n) et un jeu de $m \times n$ pièces.

Question : Existe-t-il un pavage de dimensions $m \times n$ tel que pour toute pièce, chaque bord de cette pièce est numéroté par l'opposé du bord de la pièce adjacente ?

— CONTOUR :

Instance : Un couple (m, n) et un jeu de $2m + 2n - 4$ pièces.

Question : Existe-t-il un contour de dimensions $m \times n$ tel que pour toute couple de pièces adjacentes, les bords en contact soient numérotés par l'opposé l'un de l'autre ?

— PUZZLE-ORIENTATION :

Instance : Un couple (m, n), un jeu de $m \times n$ pièces, une liste correspondant à l'orientation des pièces.

Question : Existe-t-il un placement des pièces tel que pour toute pièce du pavage obtenu, chaque bord de cette pièce soit numéroté par l'opposé du bord de la pièce adjacente en respectant l'orientation des pièces ?

— CONTOUR-ORIENTATION :

Instance : Un couple (m, n), un jeu de $2m + 2n - 4$ pièces, une liste correspondant à l'orientation des pièces.

Question : Existe-t-il un placement des pièces tel que pour toute pièce du contour obtenu, chaque bord de cette pièce soit numéroté par l'opposé du bord de la pièce adjacente ?

— PUZZLE-POSITION :

Instance : Un couple (m, n), un jeu de $m \times n$ pièces, une liste correspondant à la position des pièces dans le pavage.

Question : Existe-t-il une orientation des pièces telle que pour toute pièce du pavage obtenu, chaque bord de cette pièce est numérotée par l'opposé du bord de la pièce adjacente ?

— CONTOUR-POSITION :

Instance : Un couple (m, n), un jeu de $m \times n$ pièces, une liste correspondant à la position des pièces dans le pavage.

Question : Existe-t-il une orientation des pièces telle que toute pièce du pavage obtenu, chaque bord de cette pièce est numéroté par l'opposé du bord de la pièce adjacente ?

— SET PACKING :

Instance : Un ensemble E et un ensemble S de sous-ensembles de E.

Question : Existe-t-il un choix de n éléments de S n'admettant aucun élément commun ?

— 1-IN-3 SAT :

Instance : Une formule logique φ sous forme CNF.

Question : Existe-t-il une valuation pour laquelle exactement une des variables de chaque clause est évaluée à vraie ?

7.3 Algorithmes

7.3.1 Algorithme des k-moyennes

Algorithme 2 : Algorithme des k-moyennes

```
C ← Selectionner(puzzle, k);
changement ← vrai;
tant que changement faire
    C' ← [ ] ;
    pour p ∈ puzzle faire
        p.cluster ← PlusProche(C, p);
    fin
    pour c ∈ C faire
        C' ← C' ∪ NouveauCentre(c);
    fin
    si C' = C alors
        changement ← faux;
    fin
    C ← C';
fin
```

7.3.2 $\frac{2}{k}$-approximation de k-Set packing

Algorithme 3 : Algorithme pour k-Set packing

```
P ← [];
pour s ∈ S faire
    ajout ← vrai ;
    pour e ∈ P faire
        si s ∩ e ≠ ∅ alors
            ajout ← faux ;
        fin
    fin
    si ajout alors
        P ← P ∪ {s};
    fin
fin
tant que vrai faire
    pour (s, s') ∈ S² faire
        pour p₀ ∈ P faire
            ajout ← vrai ;
            pour p ∈ P\{p₀} faire
                si s ∩ p ≠ ∅ ou s' ∩ p ≠ ∅ alors
                    ajout ← faux ;
                fin
            fin
            si ajout alors
                P ← (P\{p₀}) ∪ {s, s'};
                fin pour;
            fin
        fin
    fin
fin
```

7.3.3 Obtention des sous-ensembles de 4 pièces valides

Algorithme 4 : Obtention des sous-ensembles de 4 pièces

```
E_{2,2} ← [];
pour (p₁ = (_, i, j, _), p₄ = (k, _, _, l)) ∈ P² faire
    p₂ ← faux;
    p₃ ← faux;
    pour p ∈ P\{p₁, p₄} faire
        si p = (_, _, k, l) alors
            p₂ ← vrai;
        fin
        si p = (j, l, _, _) alors
            p₃ ← vrai;
        fin
    fin
    si p₂ et p₃ alors
        E_{2,2} ← E_{2,2} ∪ {(p₁, p₂, p₃, p₄)};
    fin
fin
```