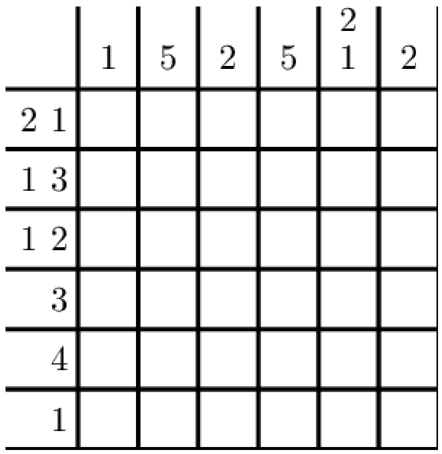
Tristan Lemoine

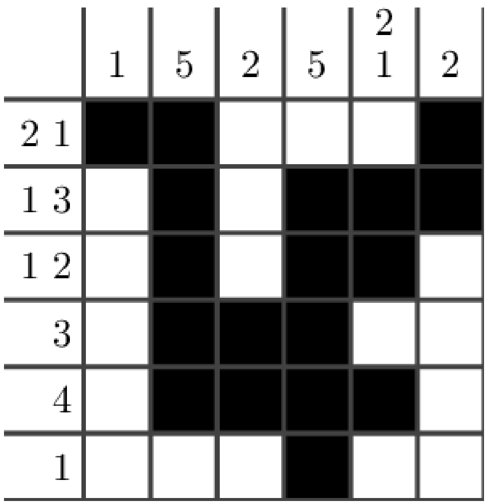
Candidat 41679

Résolution de nonograms à l’aide de différentes méthodes de programmation

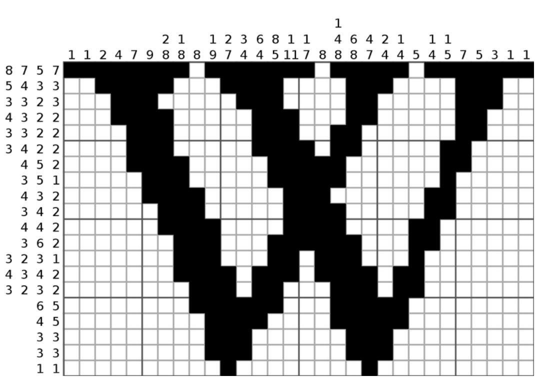
Présentation du problème

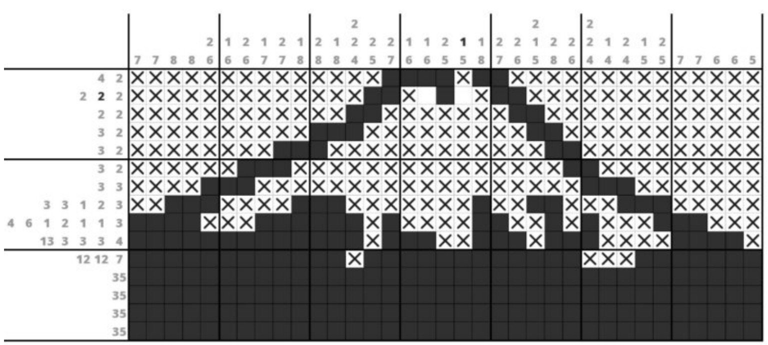


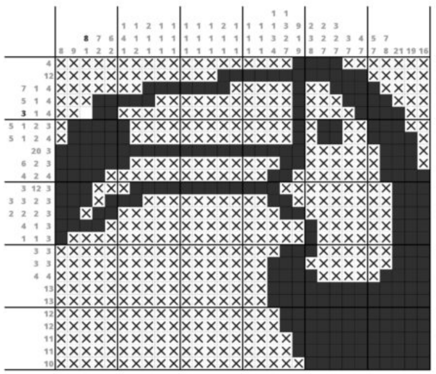
Non résolu



résolu







Images tirées de “Constructing Simple Nonograms of Varying Difficulty“ de Kees Joost Batenburg ,la page wikipedia des nonograms et la revue “Picross For A Cause” par buried-treasaure.org

Règles du jeu

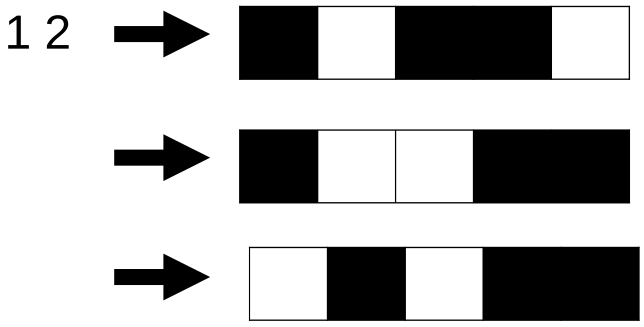
Chaque ligne contient l'ordre et la taille des blocs qui occupent la ligne

-Un bloc est un nombre succesif de blocs noirs

L'inconnue dans ce jeu est donc la position de ces blocs sur la ligne

-Deux blocs sont séparés par au moins une case blanche

Si toutes les lignes et colonnes sont satisfaites alors la grille est résolue

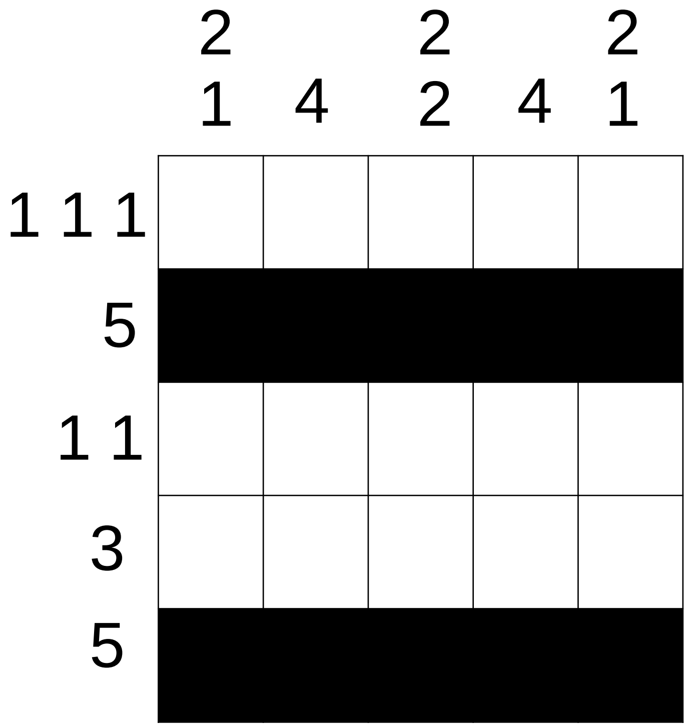


Note:Ce jeu est un problème NP-complet

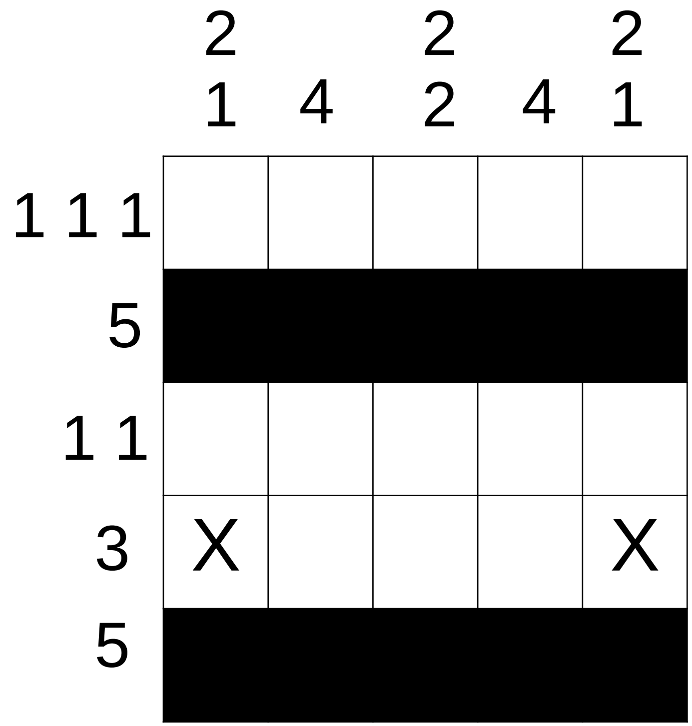
Résolution d’un nonogram

	2 1	4	2 2	4	2 1
1 1 1					
5					
1 1					
3					
5					

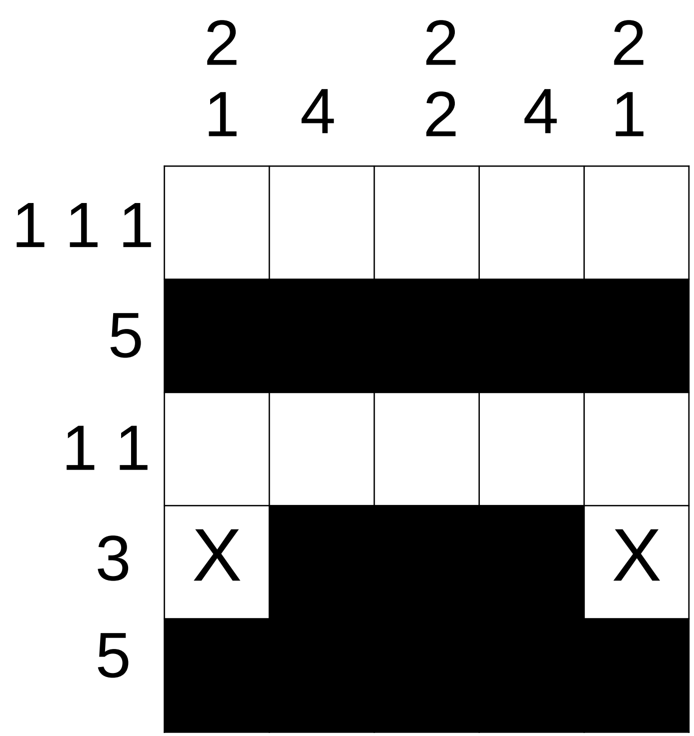
Résolution d’un nonogram



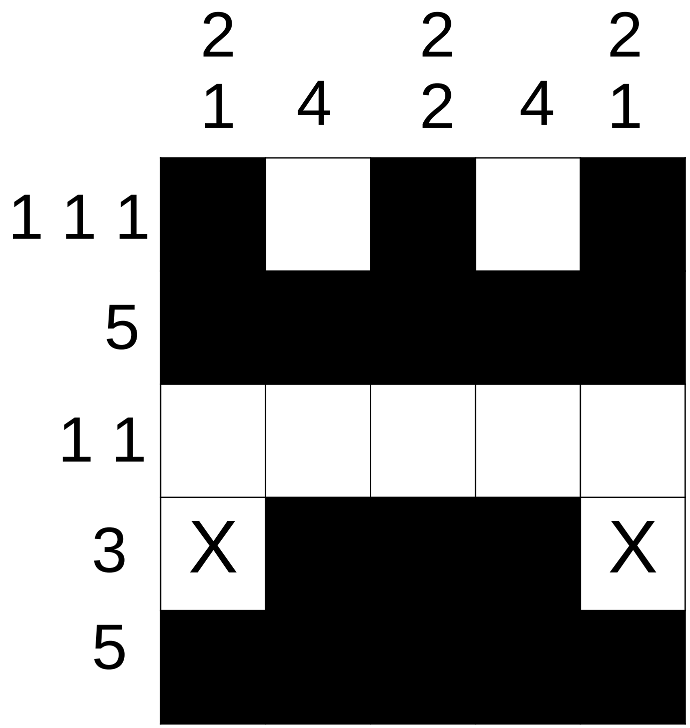
Résolution d'un nonogram



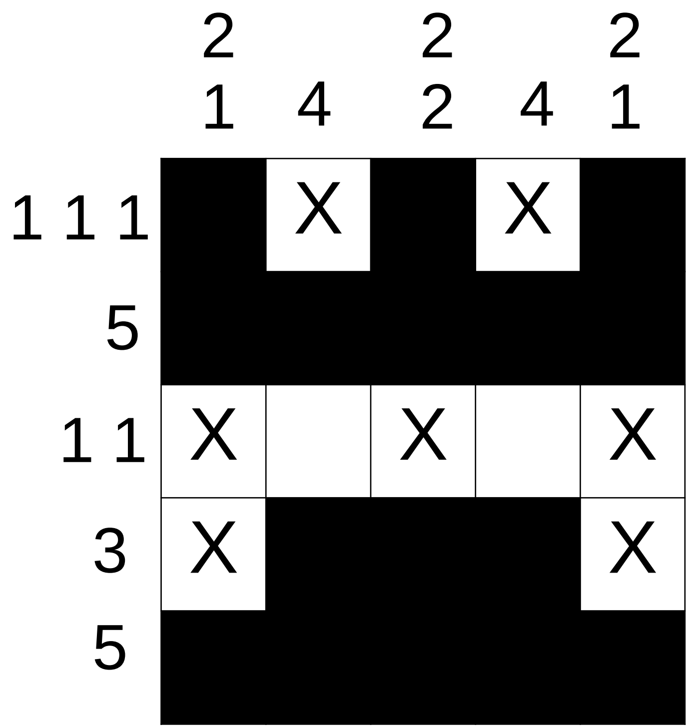
Résolution d'un nonogram



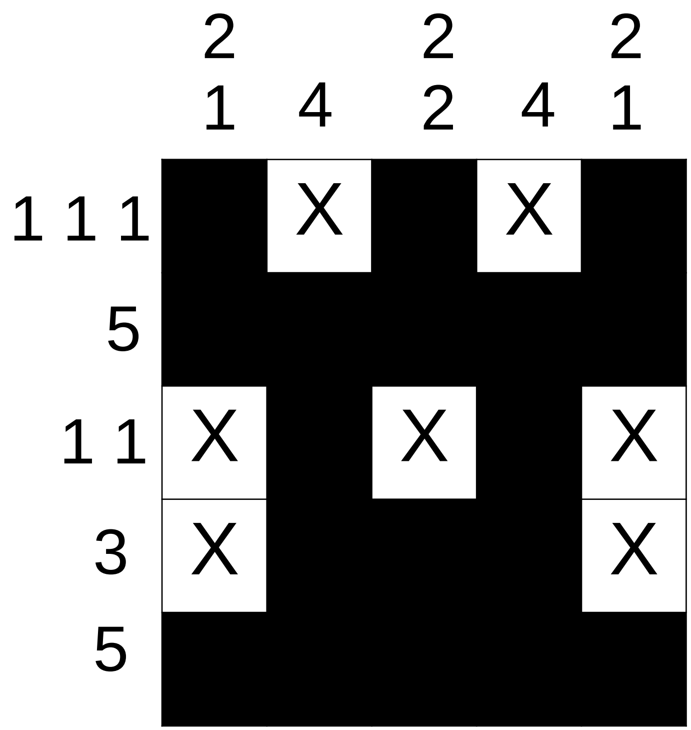
Résolution d'un nonogram



Résolution d’un nonogram



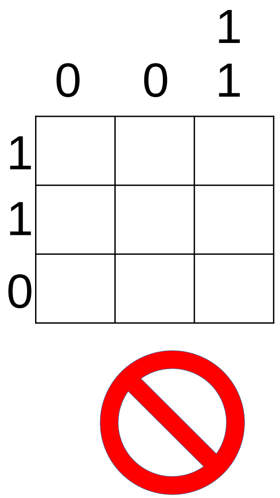
Résolution d'un nonogram



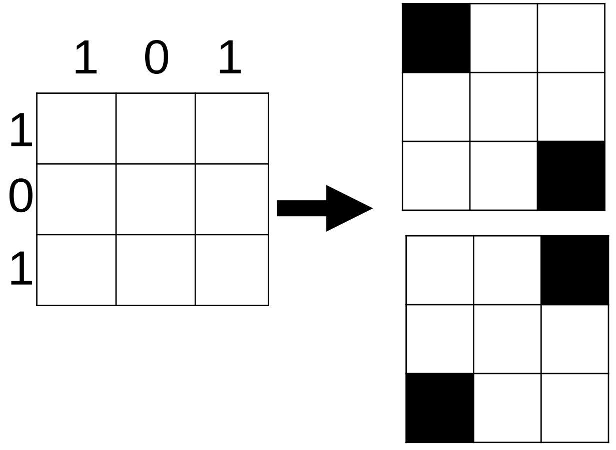
Cadre du problème

On cherche à résoudre le plus rapidement possible une grille initialment vide.

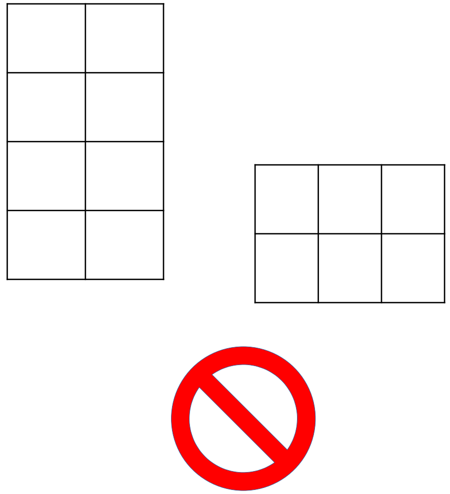
Contraintes:



Au moins une solution possible



Seule une solution est nécessaire pour considerer la grille comme résolue



La grille est carrée

La grille

Grille:

-Tableau de tableaux d’entiers

- 0 pour case vide et 1 pour case pleine

```
struct picross_grid_s {
        int size;
        int** grid;
};
```

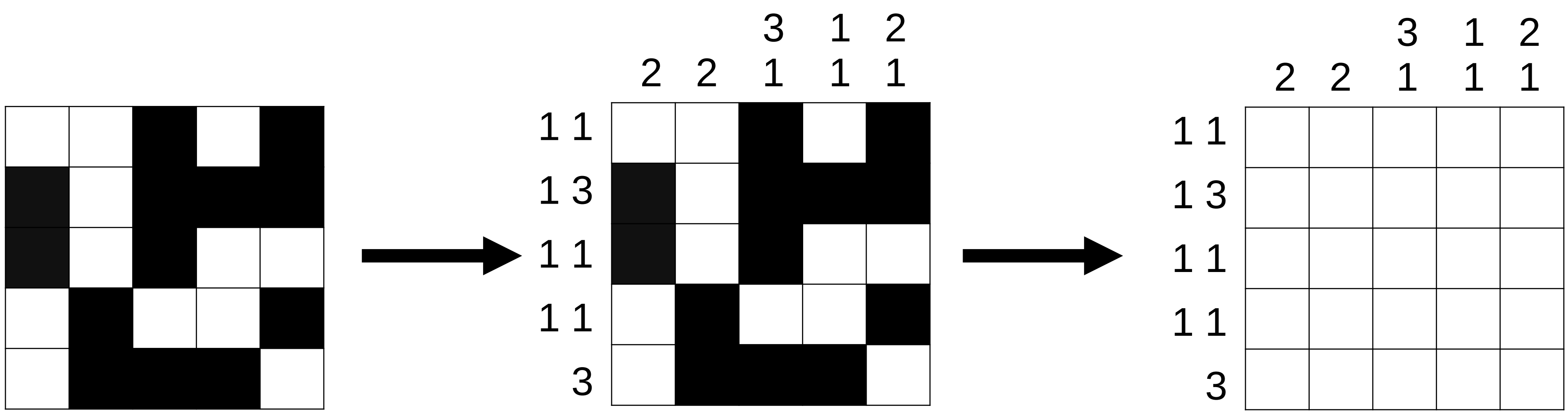
Numeros:

-Deux tableaux de listes (Lignes et colones)

-Chaque liste possède les nombres de gauche a droite ou haut en bas

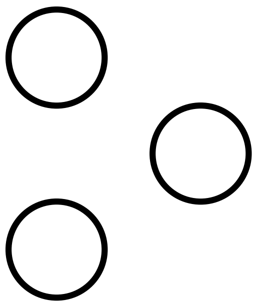
```
struct picross_numbers_s {
        liste* lig;
        liste* col;
        int size;
};
```

Generation de grille pour les algorithmes:

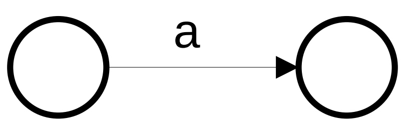


Les automates

Les automates sont construits avec:



Des états



Des relations

{0;1}

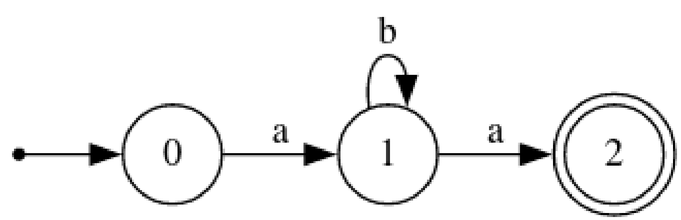
{a;b;c}

{■;□}

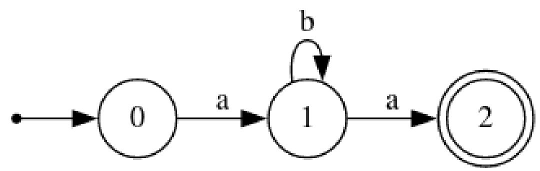
...

Un alphabet

Exemple:



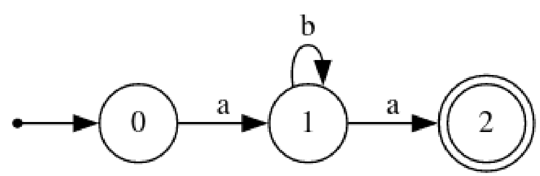
Les automates



Mots qui sont reconnus par cet automates	Mots qui ne sont pas reconnus par cet automate
-aba	-a
-aa	-abaa
-abba	-baaa
-abb...bba	...

Les automates

Si pour chaque état il n’y a qu’une destination par lettre alors l’automate est dit déterministe



	0	1	2
a	1	2	-1
b	-1	1	-1

La complexité temporelle de faire passer un mot dans un automate est linéaire en la taille du mot

```
struct automate_d_s {
        int nb_lettres;
        int nb_etats;
        int depart;
        bool* finaux;
        int** delta;
};
```

Création d’un automate pour une ligne d’un nonogram

generer_automate_ligne

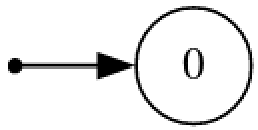
Entrée: liste de nombres L
Sortie: un automate déterministe

```
Si L est la liste vide
    → Renvoyer l’automate “zeros”
Sinon
    Créer un automate à un état de depart
    indice_etat ← 0
    Pour chaque nombre k dans L
        Connecter l’état de indice_etat a lui meme avec une relation 0
        Créer une chane de k etat relié dans un sens par une relation 1
        Connecter l’etat indice_etat au premier de cette chaine avec une relation 1
        Si k est le dernier element de la liste
            Relier le dernier element de la liste a lui-meme avec une relation 0
            Mettre ce dernier element comme état final
        Sinon
            Relier le dernier élément de la liste a un nouvel état par une relation 0
        indice_etat ← indice_etat + k +1
    →renvoyer cet automate
```

Example d’un automate

generer_automate_ligne([2, 3])

Créer un automate à un état de depart



Example d’un automate

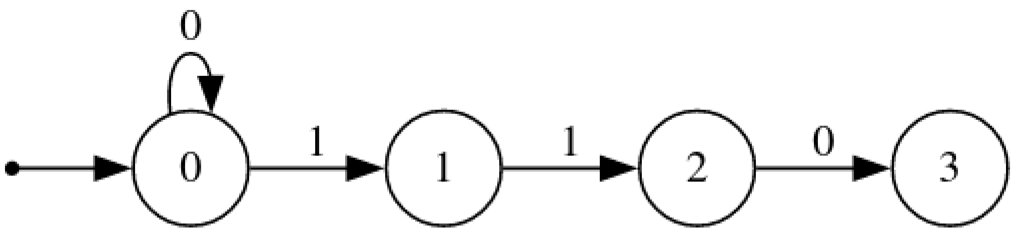
generer_automate_ligne([2, 3])

Connecter l’état de indice_etat à lui meme avec une relation 0
Créer une chane de k etat relié dans un sens par une relation 1

Example d'un automate

generer_automate_ligne([2, 3])

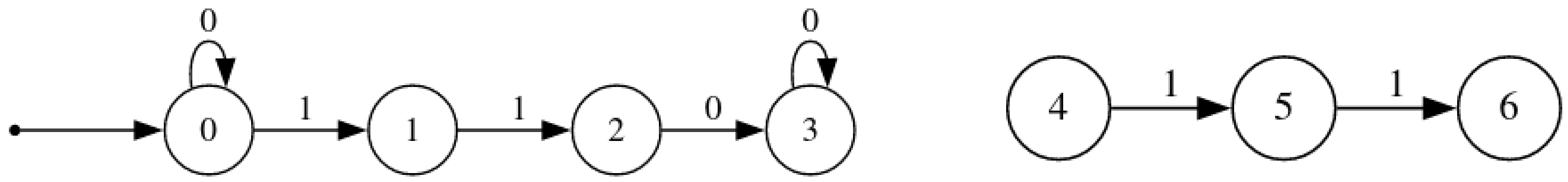
Connecter l'etat indice_etat au premier de cette chaine avec une relation 1
Relier le dernier élément de la liste a un nouvel état par une relation 0



Example d’un automate

generer_automate_ligne([2, 3])

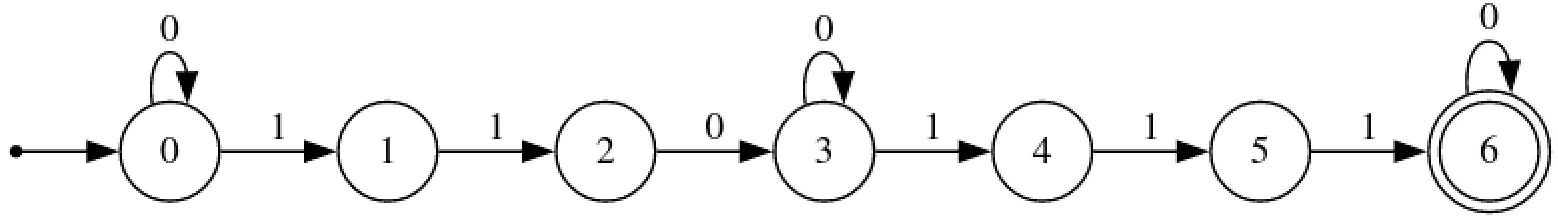
Connecter l’état de indice_etat à lui meme avec une relation 0
Créer une chane de k etat relié dans un sens par une relation 1



Example d'un automate

generer_automate_ligne([2, 3])

Connecter l'etat indice_etat au premier de cette chaine avec une relation 1
Relier le dernier element de la liste a lui-meme avec une relation 0
Mettre ce dernier element comme état final



Vérification d’une grille

est_solution_valide_total
Entrée une grille et un valideur
Sortie:Un booléen

Si toutes les lignes et colonnes sont validés par leur automates
→Renvoyer Vrai
Sinon
→ Renvoyer Faux

La complexité de cet algorithme est en $O(n^2)$

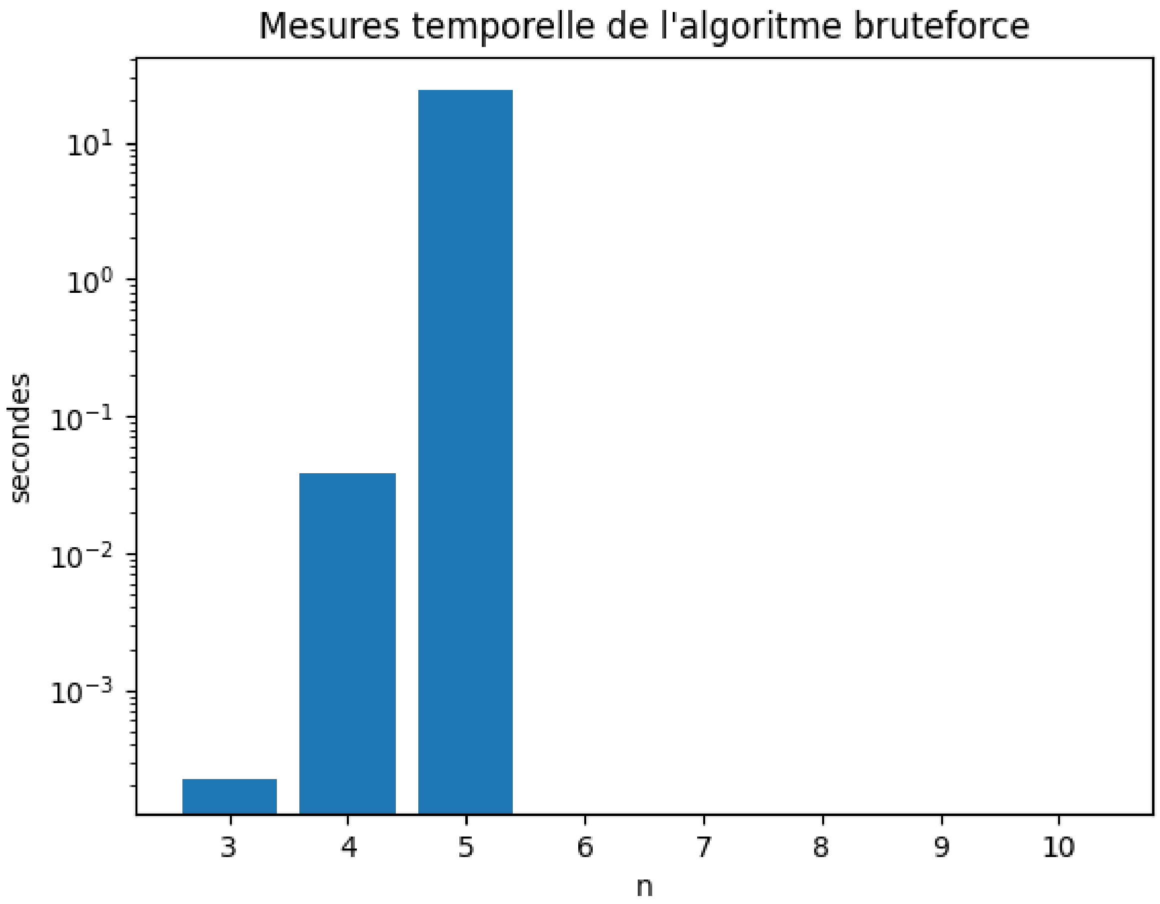
Avec n la taille de la grille

Premier algorithme

bruteforce
Entrée:Des numéros de grille
Sortie:Une grille valide

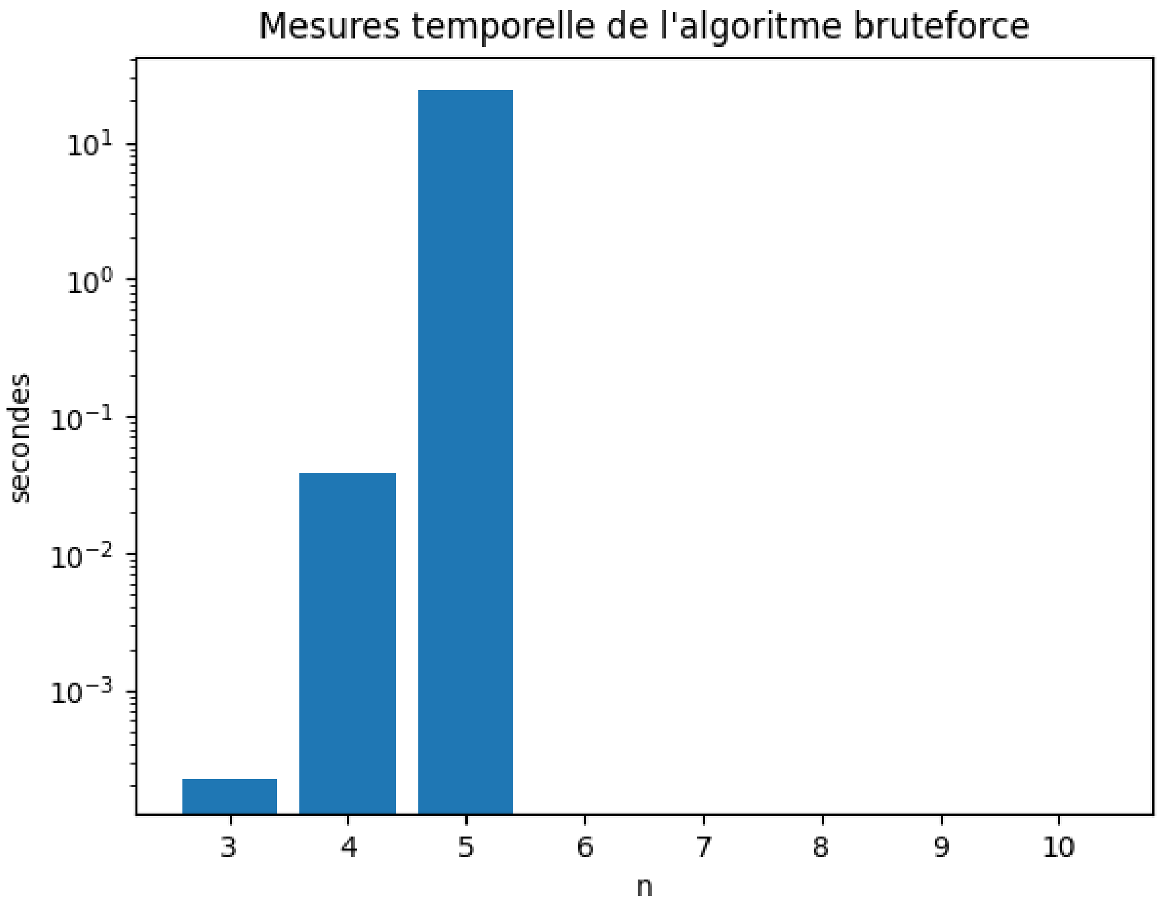
```
v ← Un valideur crée a partir des numéros de grille
Pour chaque grille g possible
        si (est_solution_valide_total(g,v) = Vrai)
                → Retourner g
```

Résultats



Mesures temporelle de l'algoritme bruteforce
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
secondes
3
4
5
6
7
8
9
10
n

Résultats

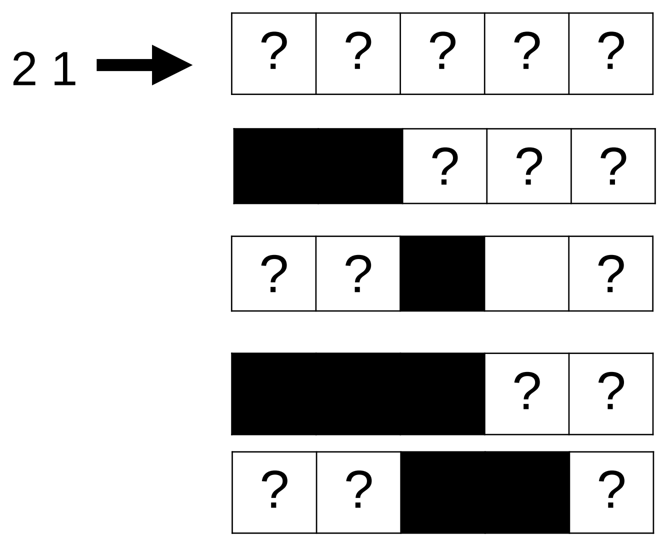


La complexité de cet algorithme est $O(2^{n2}n^2)$

Les solutions partielles

On introduit la case “inconnue” ?

Une ligne est une solution partielle si il existe une disposition des cases inconnues tel que la ligne puisse etre validée



Solution partielle

Solution partielle

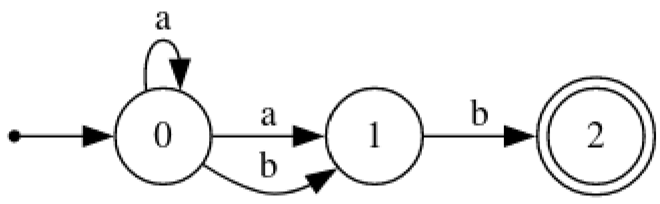
Solution partielle

Pas solution partielle

Pas solution partielle

Les automates non déterministes

Un automate qui pour au moins un état possede plusieurs relations portant le même symbole est un automate non-deterministe

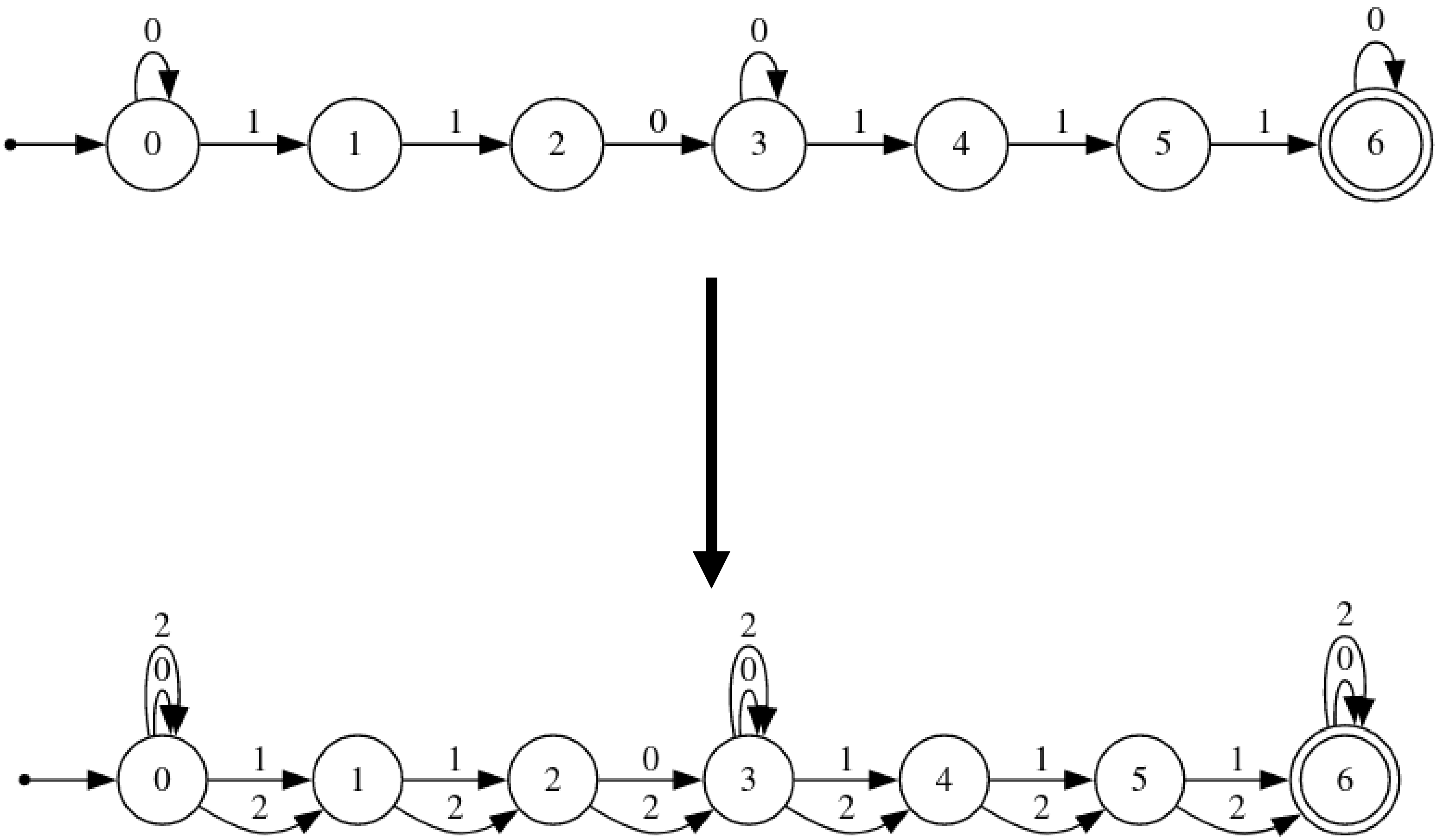


A chaque embranchement on “divise” l’état actuel pour explorer chacun des branchements

-Si un seul de ces branchement arrive a un état final alors le mot est reconnu
-Si aucun de ces branchements arrive a un état final le mot n’est pas recoonnu

La complexité pour faire passer un mot a travèrs un algorithme non deterministe est en O(Q*I)
avec
Q le nombre d’états dans l’automate
I la taille du mot d’entrée

Création de l'automate partiel pour les lignes



Determiniser l’automate?

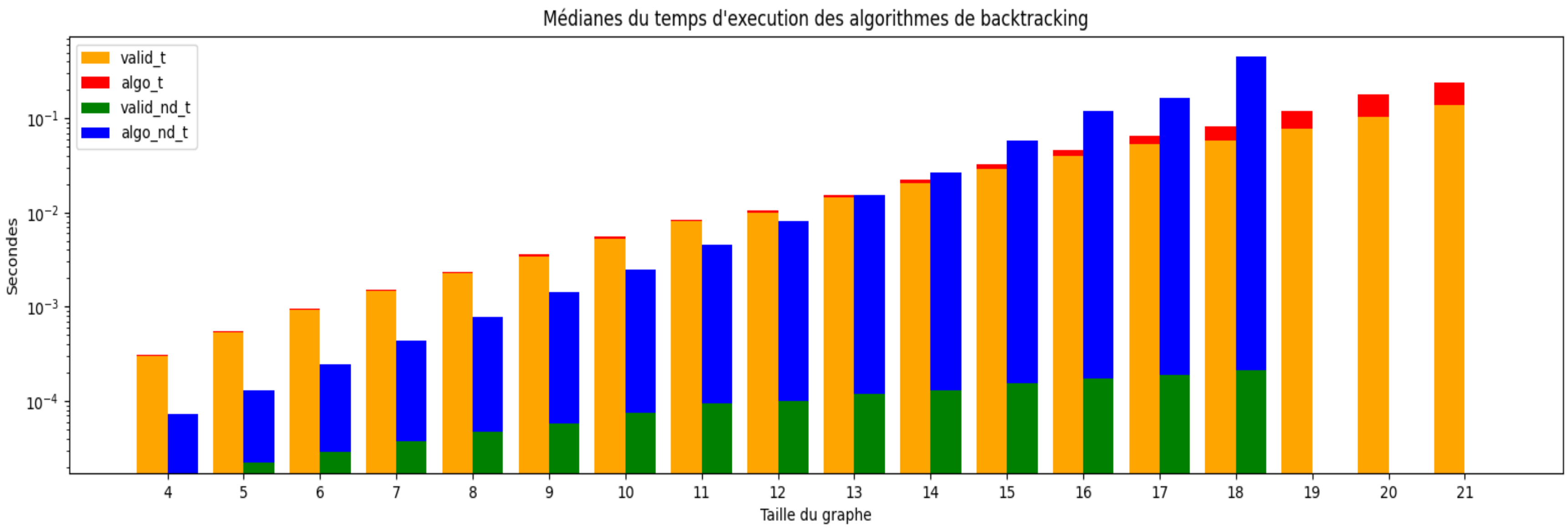
Determiniser l’automate	Laisser l’automate non-deterministe
-Creer l’automate est extremement couteux	-Creer l’automate est tres rapide
-Passer un mot à travers l’automate est rapide	-Passer un mot a travres l’automate est plus long

Algorithme de backtracking

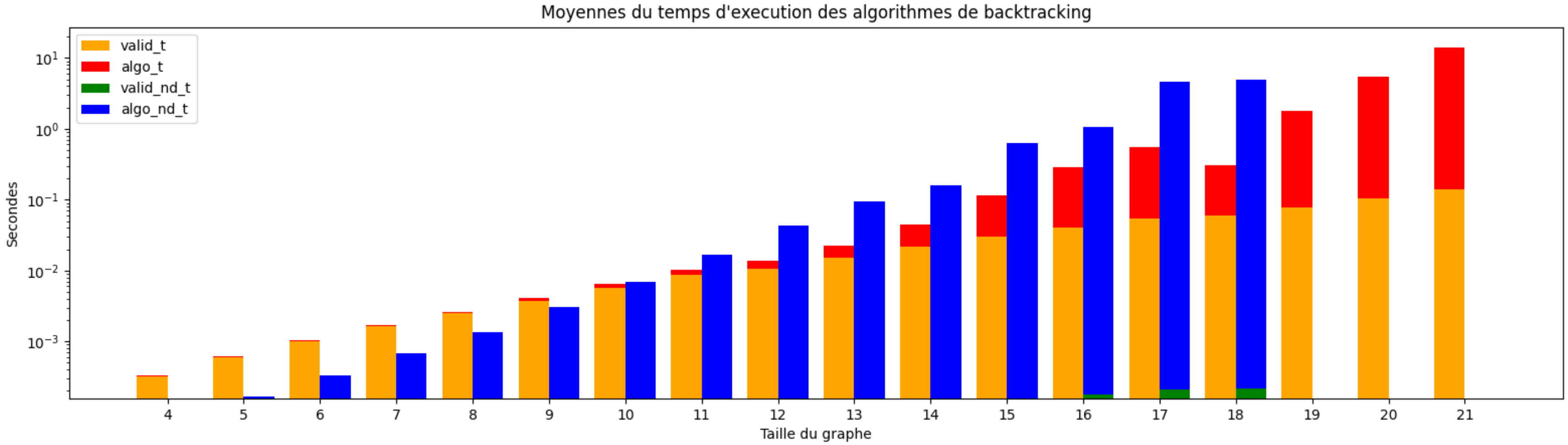
backtrack
Entrée:Une grille,un valideur,les coordonées (I,j) d’une case c
Sortie:Un booléen
Effet:La grille mise en entrée est changée en grille valide

```
Si c est la derniere case:
    colorier la case c en blanc
    si la ligne et la colonne de c sont reconnu par leur automates:
        →Retourner Vrai
    colorier c en noir
    si la ligne et la colonne de c sont reconnu par leur automates:
        →Retourner Vrai
    colorier c en inconnu
        → retourner Faux
Sinon
    colorier la case c en blanc
    si la ligne et la colonne de c sont reconnu par leur automates:
        backtrack a la case suivante
    colorier c en noir
    si la ligne et la colonne de c sont reconnu par leur automates:
        backtrack a la case suivante
    colorier c en inconnu
        → retourner Faux
```

Algorithme de backtracking



Algorithme de backtracking



Règles logiques

Les regles logiques sont un moyen aux algorithmes de backtracking d'avoir moins de grilles a parcourir

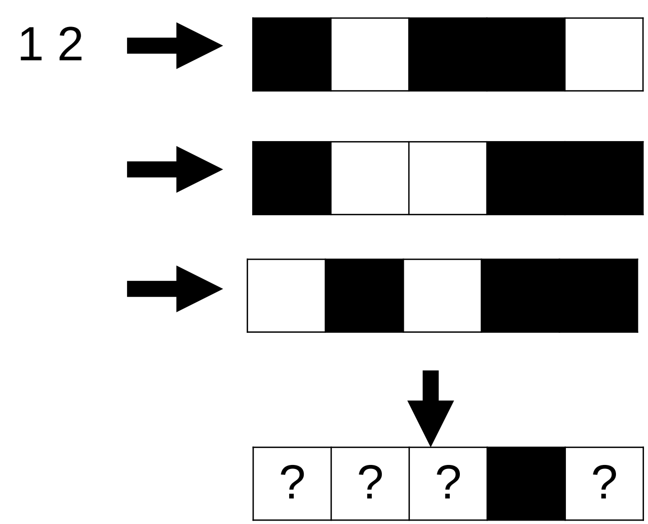
Première regle:
Si une ligne possede comme numéro 0 alors toutes ces cases sont blanches

Règles logiques

Les regles logiques sont un moyen aux algorithmes de backtracking d'avoir moins de grilles a parcourir

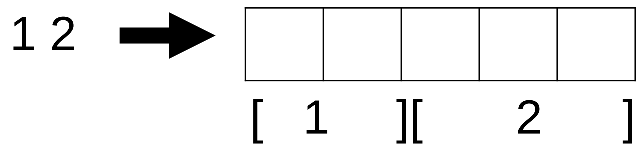
Première regle:
Si une ligne possede comme numéro 0 alors toutes ces cases sont blanches

Deuxieme regle:
Si une case est noire dans chaque positions valide de la ligne alors celle ci est noire



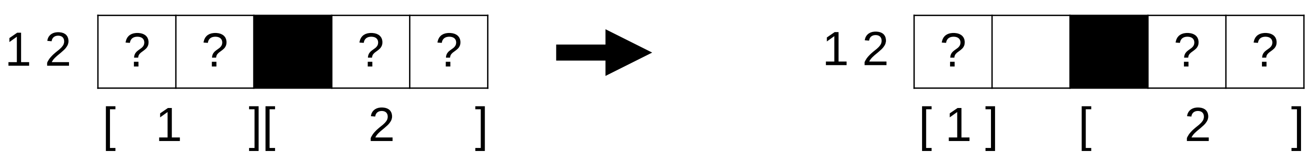
Règles logiques

La portée d'un bloc est l'intervalle sur lequel le bloc peut etre placé

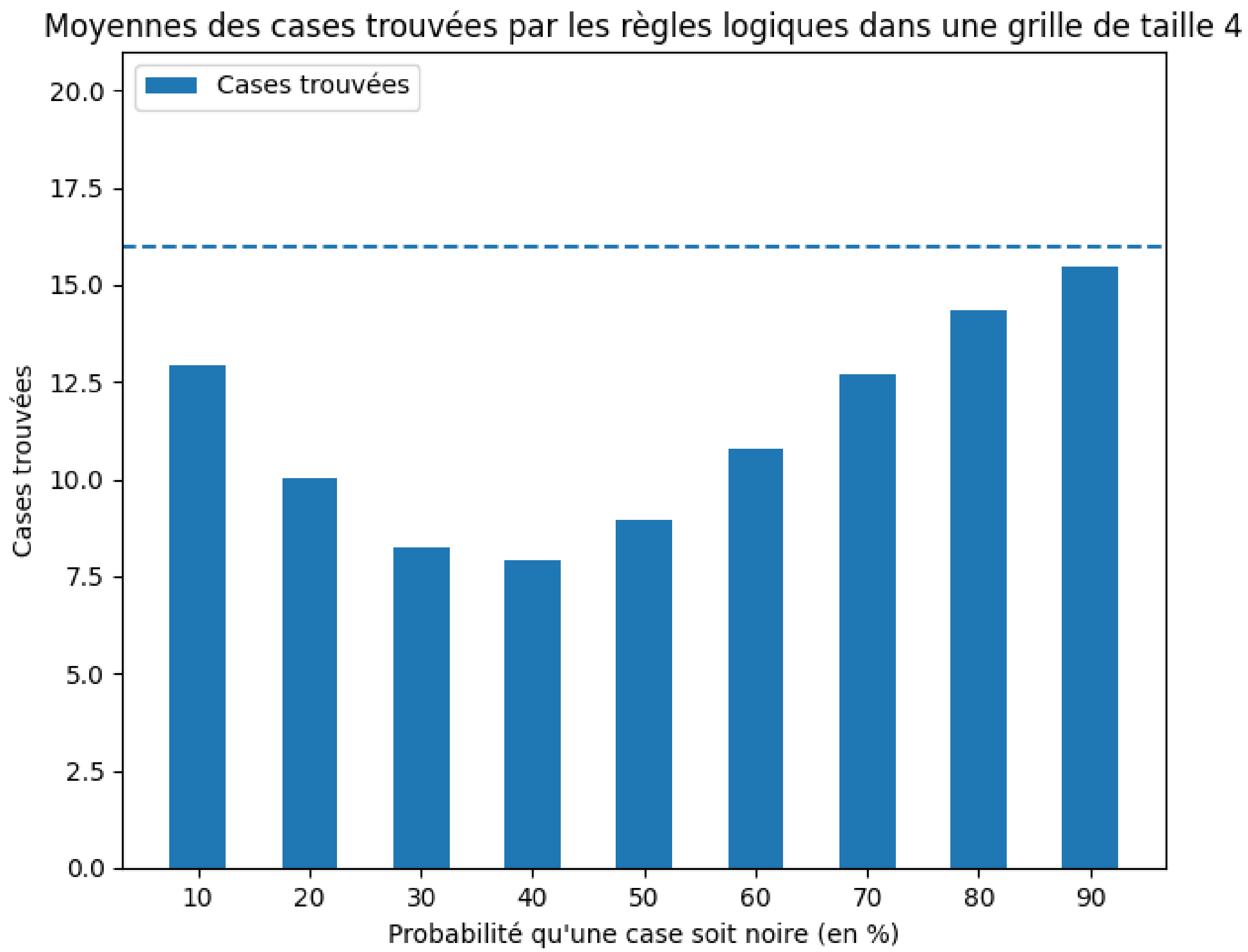


Troisieme regle:
Si un bloc noir est a coté de l'extememité d'une portée d'un bloc,on peut réduire cette portée

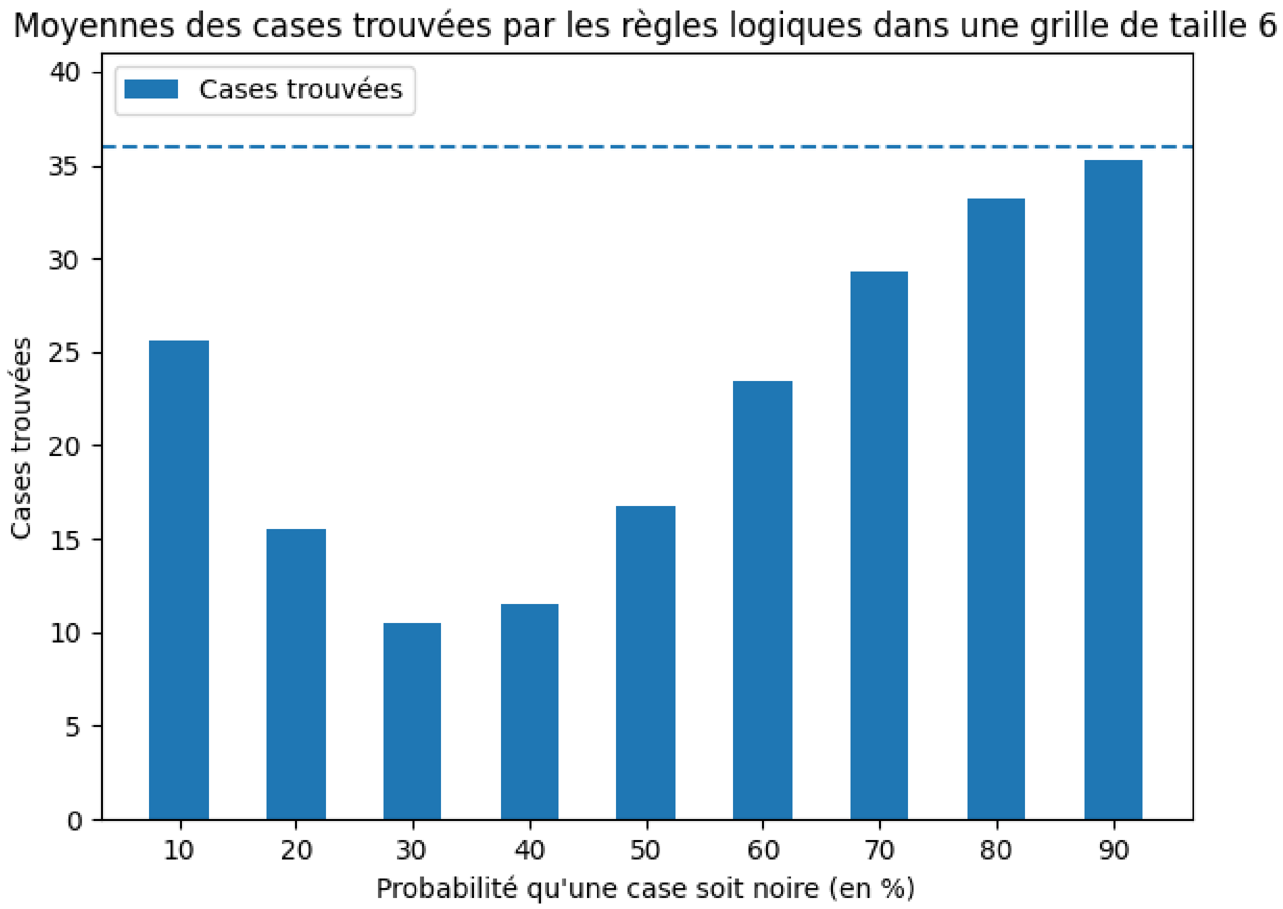
Quatrieme regle:
Si une case n'est dans aucune portée, cette case est blanche



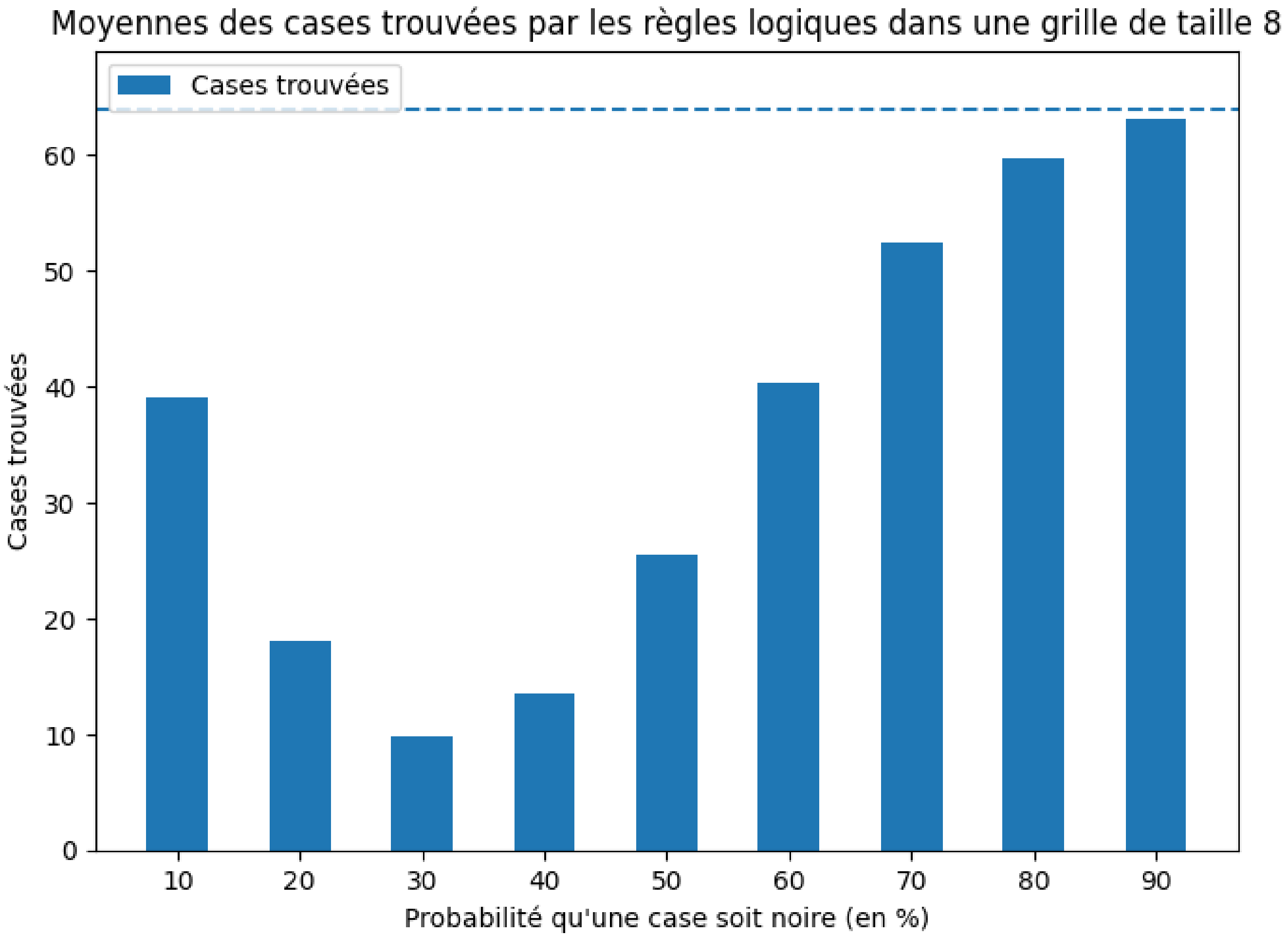
Règles logiques



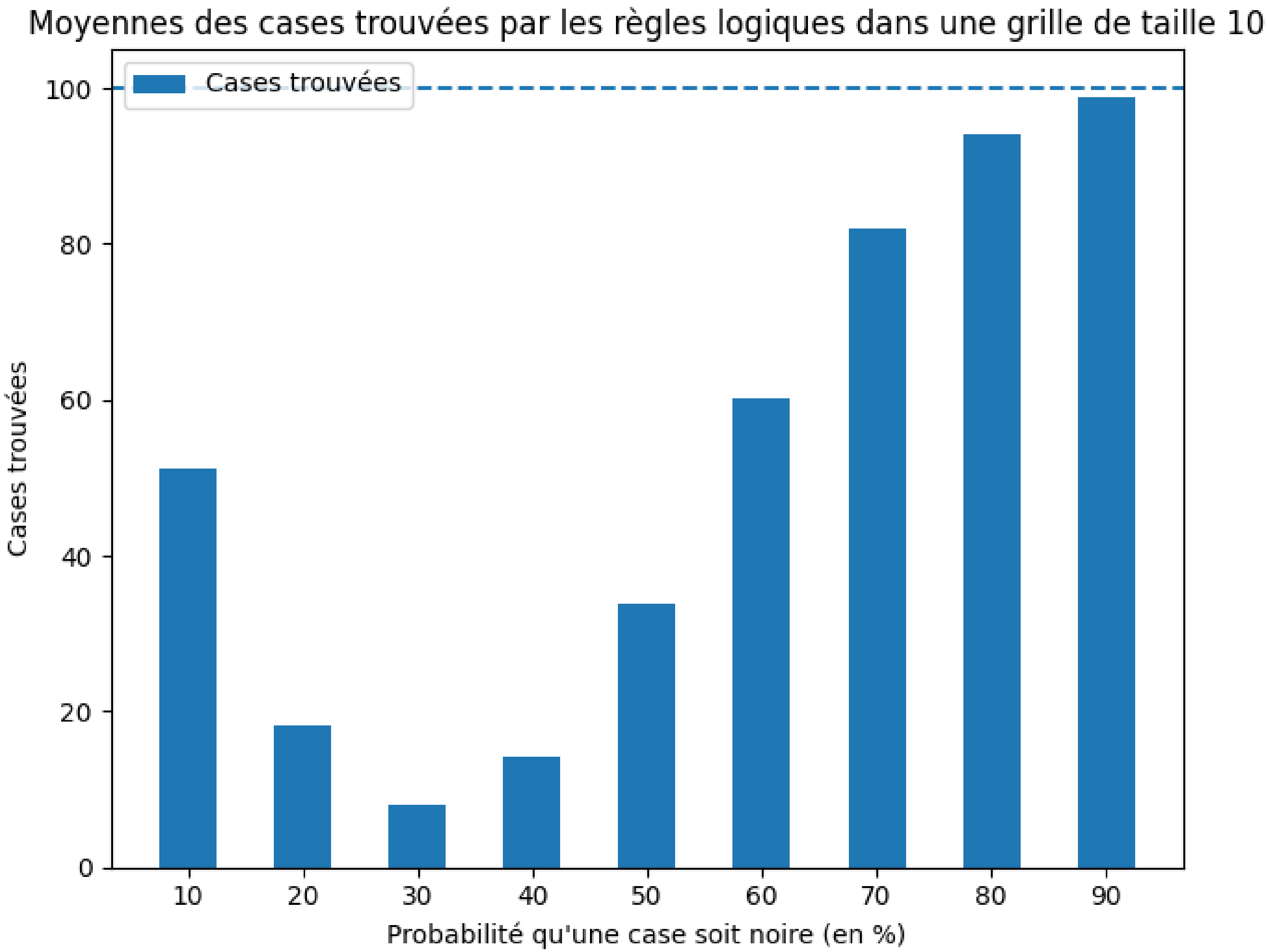
Règles logiques



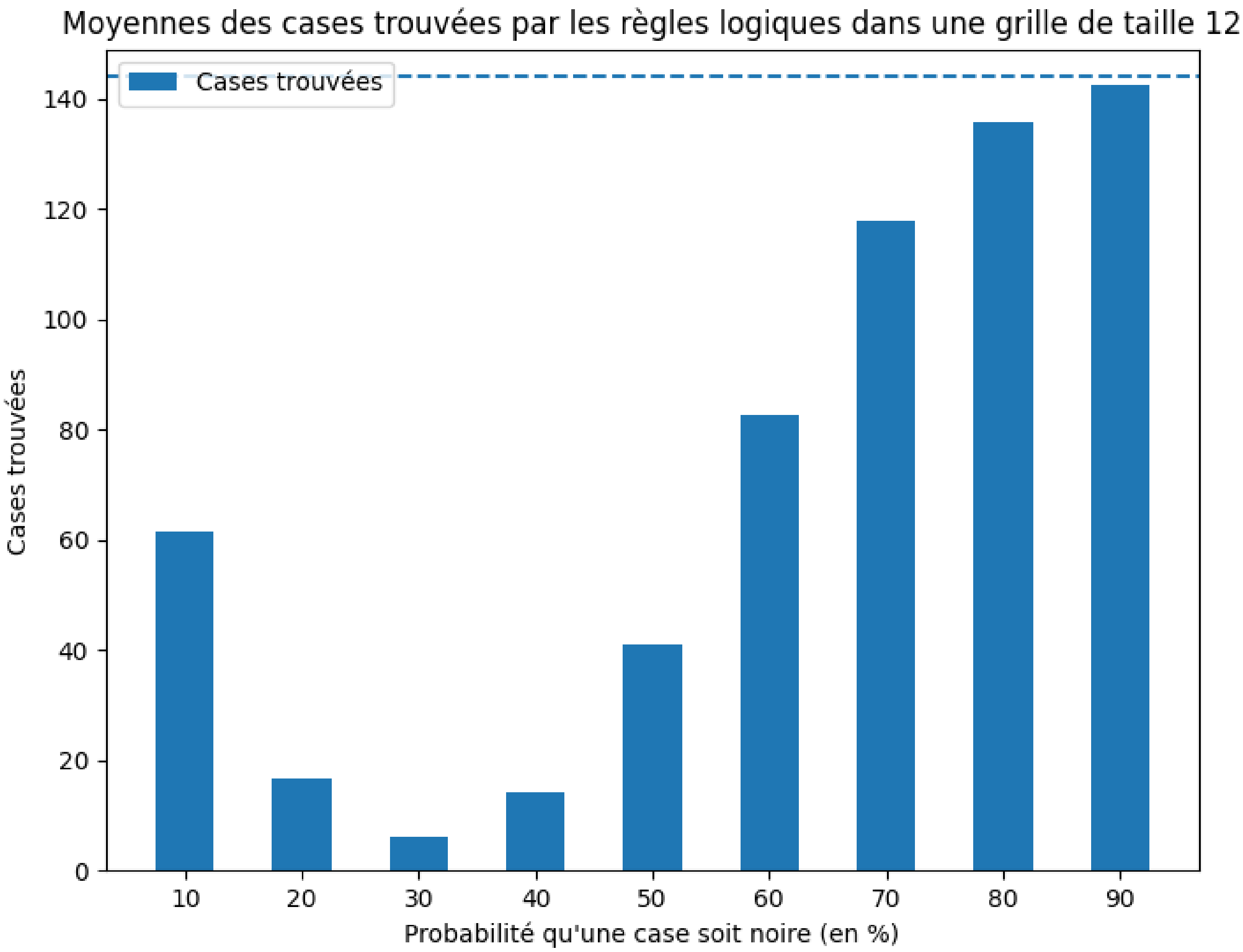
Règles logiques



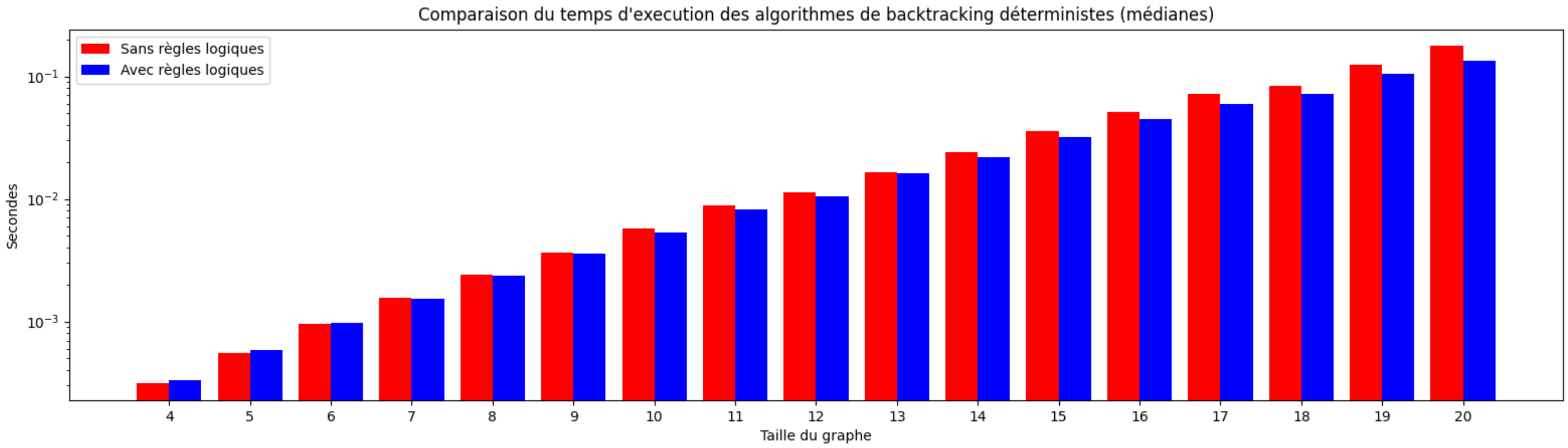
Règles logiques



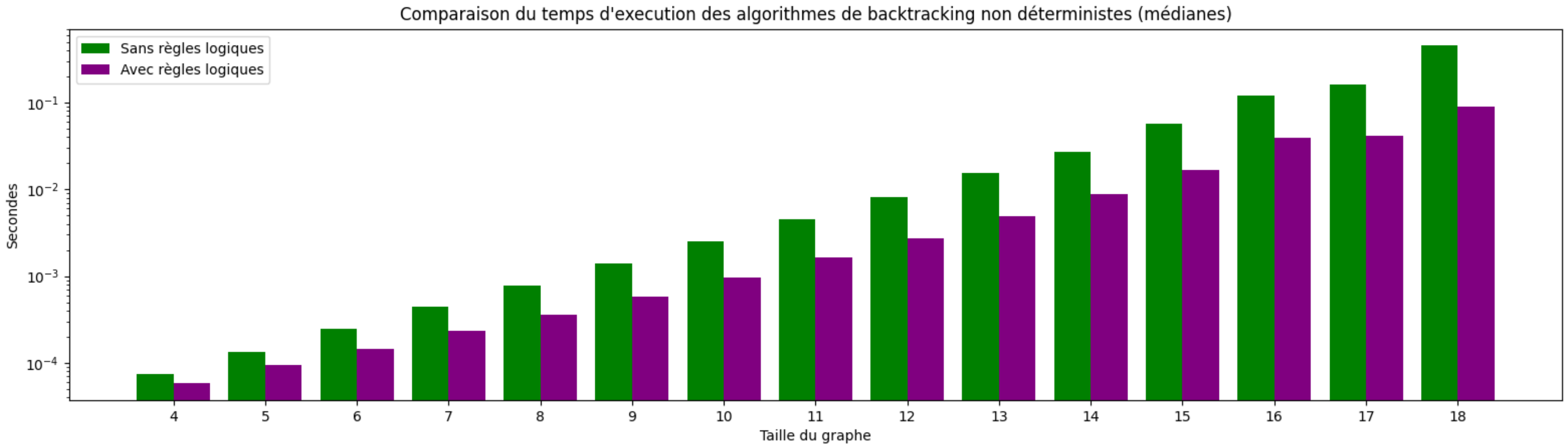
Règles logiques



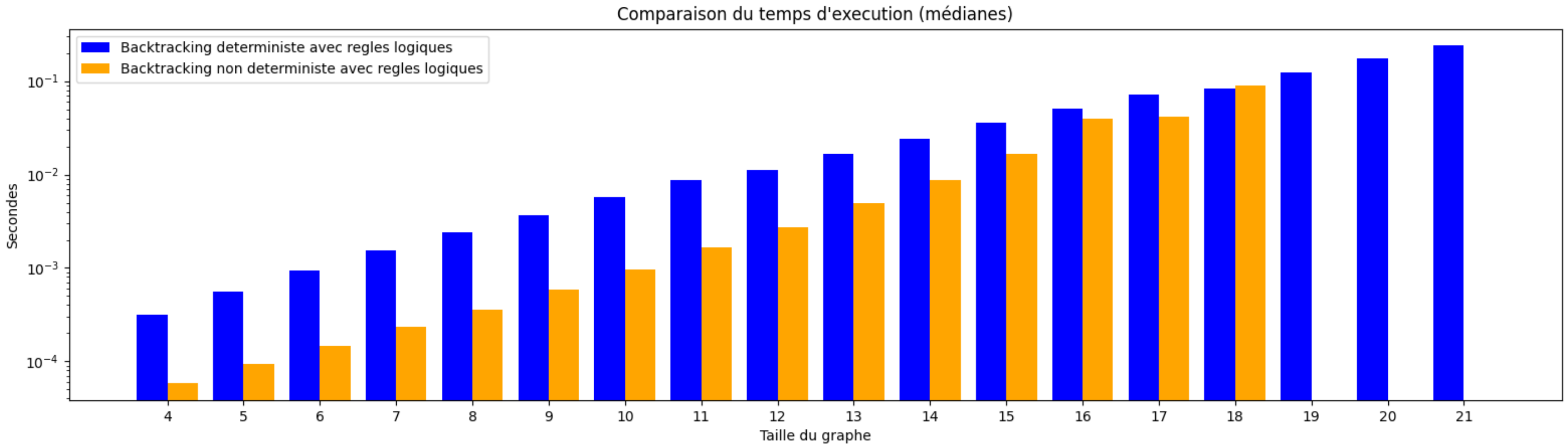
Conclusions



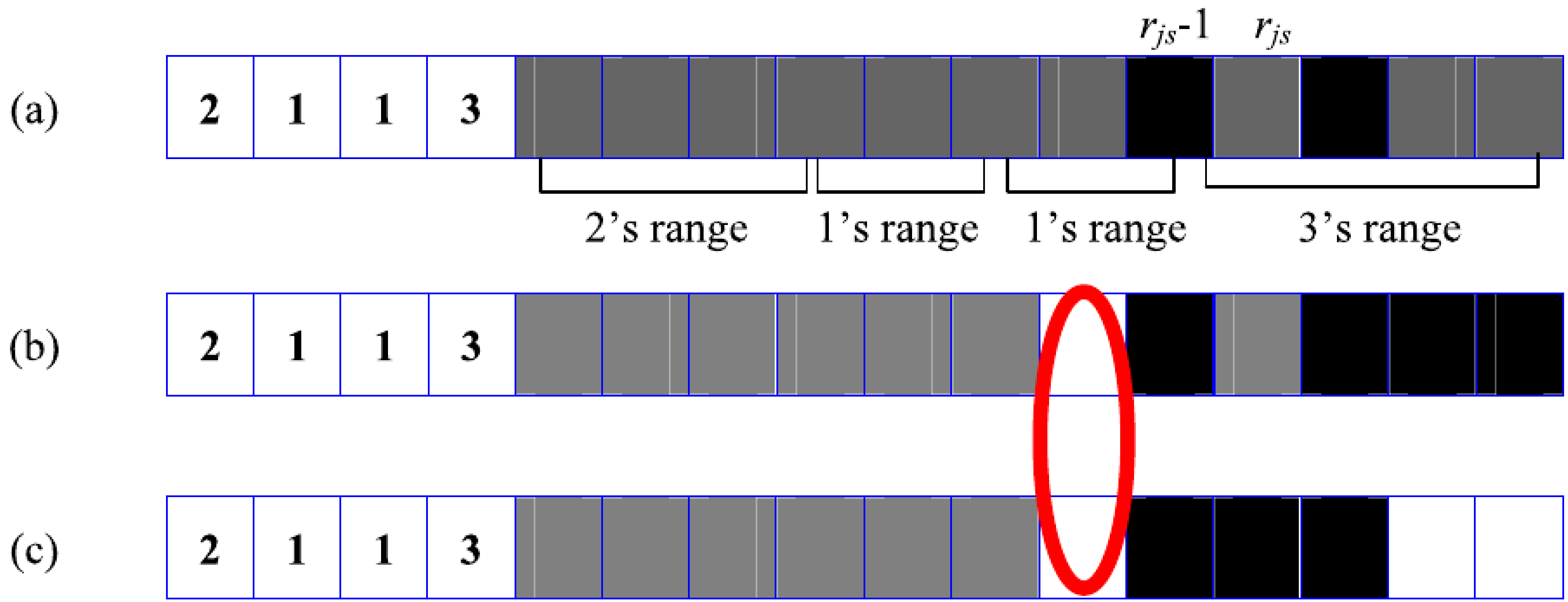
Conclusions



Conclusions



Annexe



Tirée de “An efficient algorithm for solving nonograms”