Filières MP/MPI - ENS de Paris-Saclay, Lyon, Rennes et Paris - Session 2025
Page de garde du rapport de TIPE

Attention: Le rapport doit être réalisé par l'étudiant(e).
Si le rapport résulte d'une collaboration, elle doit être clairement annoncée.

NOM : LANDREAU	Prénoms : Félix Fernand Marc Alexandre
Classe : MPI*	
Lycée : Clemenceau	Numéro de candidat : 13785
Ville : Nantes	

Concours auxquels vous êtes admissible (les indiquer par une croix) :

Banque MP

ENS Paris-Saclay	MP - Option MP	
	MP - Option MI	
ENS de Lyon	Math - Option M-MP	
	Math - Option M-MI	
	Informatique	
ENS Rennes	MP - Option MP	
	MP - Option MI	
ENS Paris	MP - Option P	
	MP - Option I	

Banque MPI

ENS Paris-Saclay	Option Math	
	Option Info	X
ENS de Lyon	Math MPI	
	Info MPI	X
ENS Rennes	Info MPI	X
ENS Paris	Option Math	X
	Option Info	

Matière dominante du TIPE (la sélectionner d'une croix inscrite dans la case correspondante) :

Informatique	X	Mathématiques		Physique	

Titre du TIPE : Mise en place de méthodes d'évaluation de la difficulté d'une grille de sudoku

Nombre de pages (à indiquer dans les cases ci-dessous) :

Texte	5	Illustration	4	Bibliographie	1

Attention, les illustrations doivent figurer dans le corps du texte et non en fin du document !

Résumé ou descriptif succinct du TIPE (6 lignes, maximum) :

Ce TIPE a pour but d'établir pour les grilles de sudoku une échelle de difficulté correspondant au ressenti humain. Pour cela, il met en place plusieurs approches : d'abord, il teste des approches virtuelles simples. Ensuite, il essaye de transcrire le raisonnement humain en un algorithme de résolution, dont le déroulement servira de base à l'évaluation de la difficulté. Enfin, il réduit le sudoku au problème SAT et explore son lien avec la difficulté ressentie des problèmes traduits.

A Nantes

Le 4 juin 2025

Signature du (de la) candidat(e)

Félix

Signature du professeur responsable de la classe préparatoire dans la discipline

Mathieu Journault

Cachet de l'établissement

LYCÉE CLEMENCEAU ★ NANTES ★ LE PROVISEUR ADJOINT

La signature du professeur responsable et le tampon de l'établissement ne sont pas indispensables pour les candidats libres (hors CPGE)

Mise en place de méthodes d'évaluation de la difficulté d'une grille de sudoku.

Félix Landreau

7 juin 2025

Résumé

Ce TIPE a pour but d'établir pour les grilles de sudoku une échelle de difficulté correspondant au ressenti humain. Pour ce faire, il met en place plusieurs approches : d'abord, il teste des approches visuelles simples. Ensuite, il essaye de transcrire le raisonnement humain en un algorithme de résolution, dont le déroulement servira de base à l'évaluation de la difficulté. Enfin, il réduit le problème du sudoku au problème de satisfiabilité booléenne SAT et explore son lien avec la difficulté ressentie des problèmes traduits.

1 Introduction

Le sudoku est un jeu de grille très répandu, qui consiste à remplir une grille carrée de 9×9 cases en plaçant les chiffres de 1 à 9 une fois chacun dans chaque ligne, colonne et zone carrée de 3 cases de côté. Il y a une littérature scientifique abondante concernant les algorithmes de résolution et de génération de grilles. Cependant, peu de travaux scientifiques ont été réalisés sur les méthodes d'évaluation de la difficulté des grilles [1] [2] [3], et il n'existe aucune échelle de difficulté commune adaptée aux humains. Ainsi, chaque éditeur a recours à sa propre méthode d'évaluation, qui passe souvent par la résolution de la grille par un humain. Cette dernière méthode ne peut cependant pas servir de base à une échelle de difficulté car l'évaluation finale dépend du jugement de l'évaluateur. Il est donc intéressant d'établir d'autres méthodes d'évaluation plus systématiques.

L'enjeu ici est la transformation d'un ressenti de difficulté humain en des critères mesurables, afin d'automatiser le processus d'évaluation et de définir une échelle de difficulté unique.

Durant ce TIPE, j'ai testé mes algorithmes évaluant la difficulté d'un sudoku sur la base de données issue d'une étude décrite dans l'article [4] (344 grilles). Les données ont été collectées auprès de nombreux joueurs à l'aide d'une application mobile, sous la forme d'un temps de résolution et d'un taux d'abandon, desquels est déduite une valeur mesurant la difficulté. J'ai aussi eu recours à la base [5], pour avoir une plus grande diversité de grilles (plusieurs milliers de grilles, de difficultés plus variées).

2 Critères visuels

2.1 Première approche naturelle

Lorsque l'on souhaite évaluer la difficulté d'une grille de sudoku, le critère le plus naturel est le nombre de chiffres donnés initialement (indices). Malheureusement, suite à des tests, ce critère s'avère très peu corrélé avec la difficulté réelle (voir Figure 1). On observe que le nombre d'indices ne diminue pas forcément quand la difficulté du sudoku croît.

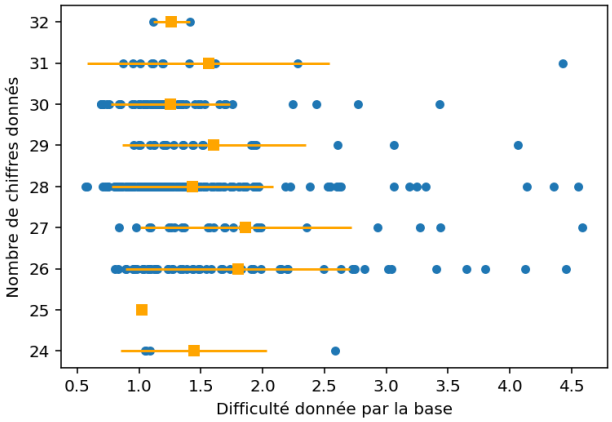


FIGURE 1 – Nombre d'indices en fonction de la difficulté donnée

J'ai alors voulu savoir si d'autres critères relativement simples, voire perceptibles au premier coup d'œil, et ne nécessitant pas la résolution de la grille étaient plus pertinents pour l'évaluation des sudokus.

2.2 Autres critères traités

Trois critères de nature statistique ont été étudiés dans cette partie :

1. Le nombre total de candidats, c'est-à-dire la somme pour chaque case du nombre de chiffres qui ne sont pas directement éliminés par les règles (voir Figure 2)

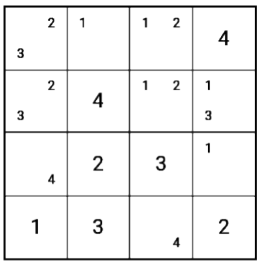


FIGURE 2 – Critère 1 : nombre total de candidats. Ce sudoku est en format 2×2 pour être plus lisible, mais les sudokus étudiés sont en 3×3

2. La répartition "géographique" des chiffres donnés. On note pour cela par colonne le nombre x_i de chiffres présents, et on calcule l'écart-type de ces valeurs. On fait de même sur les lignes et carrés de 3×3 cases (voir Figure 3).

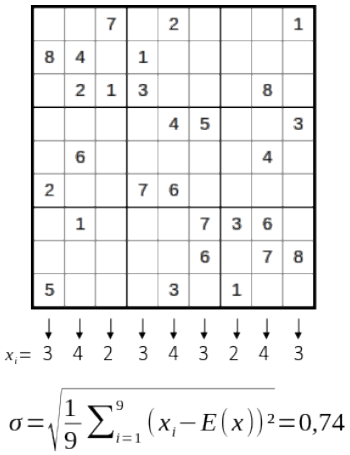


FIGURE 3 – Écart-type sur les lignes

3. La répartition entre valeurs des chiffres donnés suivant leur fréquence dans la grille, calculée par un écart-type similaire.

2.3 Résultats

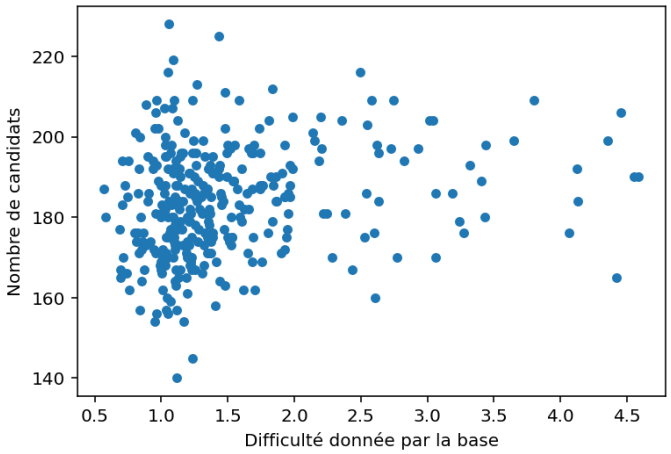


FIGURE 4 – Résultats du critère 1 (nombre de candidats)

Le critère 1 (nombre de candidats) ne convient pas pour noter une grille. En effet, on observe sur la figure 4 que le nombre de candidats n'augmente pas forcément avec la difficulté. Les résultats des deux autres critères sont similaires, ils sont disponibles en annexe en figure 11.

Ces résultats étaient en fait assez prévisibles : en effet, mon approche était d'essayer de valider des critères de difficulté intuités par des observations personnelles, sans chercher à comprendre ce qui génère la difficulté chez un humain. De plus, ces méthodes se limitent à la grille initiale, et pas aux étapes de raisonnement.

3 Évaluation par résolution

3.1 Calcul de l'utilisation des techniques

La résolution d'une grille de sudoku par un humain s'appuie sur des techniques de déduction logique propres au sudoku. Ces techniques sont des étapes de raisonnement qui permettent de remplir une case ou d'éliminer des candidats. Par exemple, la technique du singleton caché (Figure 5) traite le cas où un chiffre ne peut être placé que dans une seule case dans une ligne, colonne ou bloc de 3×3 cases.

		7		2				1
8	4		1					
	2	1	3				8	
				4	5			3
	6						4	
2			7	6				
	1				7	3	6	
					6		7	8
5	7			3		1		

FIGURE 5 – Exemple : technique du singleton caché appliquée dans le carré 3×3 en bas à gauche

C'est pourquoi, pour évaluer la difficulté d'une grille, il m'a semblé pertinent de déterminer quelles techniques étaient nécessaires à sa résolution.

J'ai donc implémenté en langage C une douzaine de techniques humaines de déduction classiques (extraits de code en figure 13 en annexe, 3000 lignes de code au total). On ordonne ces techniques par difficulté croissante pour un humain (modifiable par la suite). J'applique ensuite ces techniques

une par une par difficulté croissante à la grille à résoudre, en revenant à la première technique dès qu'une technique réussit : cela garantit que l'algorithme utilise les techniques de difficulté minimale, sous l'hypothèse que les techniques soient fournies par ordre croissant de difficulté (Figure 12 en annexe).

3.2 Obtention d'un score

Une fois obtenues les techniques nécessaires à la résolution d'une grille, il s'agit d'en déduire un score de difficulté. Dans un premier temps, j'essaie de définir un score linéaire en le nombre d'utilisations n_i de chaque technique pour chaque sudoku S :

$$S \mapsto (n_1, \dots, n_p) \mapsto \sum_{i=1}^{p} n_i \alpha_i$$

Chaque technique i se voit alors attribuer un poids α_i. Mon programme C me permet de calculer pour chaque sudoku les n_i, et la base de données [4] me donne la difficulté associée : je peux donc déterminer ces poids selon une méthode analytique des moindres carrés (programme Python).

3.3 Résultats

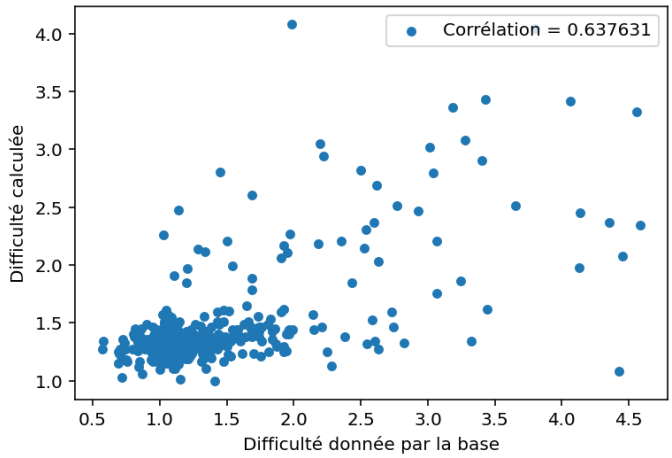


FIGURE 6 – Difficulté calculée en fonction de celle donnée par la base [4]

L'entraînement et le test ont été réalisés sur 150 sudokus chacun. Les coefficients correspondants aux différentes techniques sont :

Dernière case vide	Singleton nu	Singleton caché	Paire pointante	Paire nue	Paire cachée	Autres
0.12	0.07	0.16	-0.02	0.27	0.44	Non fiable

TABLE 1 – Poids des techniques

La difficulté calculée est assez corrélée à celle donnée par la base (Figure 6), mais l'évaluation reste imprécise. En particulier, les grilles faciles ne sont pas départagées. Cet écart à la réalité peut partiellement s'expliquer par une incertitude liée à la base de données. Seuls les premiers coefficients sont fiables, car la base de données [4] manque de grilles plus difficiles.

Afin de pouvoir évaluer un plus large spectre de techniques, j'ai utilisé une seconde base de données [5], dont le processus d'évaluation repose sur un autre procédé algorithmique. Les difficultés sont des nombres entiers dans cette base. Les résultats sont en figure 7.

Les coefficients donnés (Table 2) par l'algorithme précédent ne sont pas croissants, ce qui semble invalider l'hypothèse que les techniques sont fournies par ordre croissant de difficulté.

Cependant, après plusieurs essais, il s'est avéré qu'aucun ordre ne permettait d'assurer cette propriété (les coefficients changeant avec l'ordre).

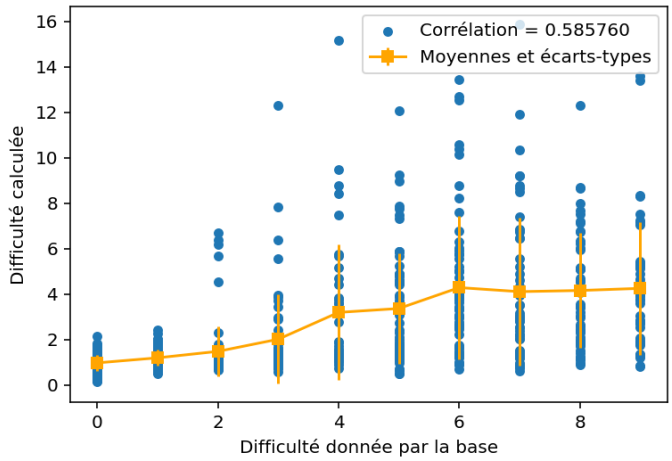


FIGURE 7 – Difficulté calculée en fonction de celle donnée par la base [5]

Dernière case vide	Singleton Nu	Singleton caché	Paire pointante	Paire nue	Paire cachée
-0.017	-0.027	0.010	0.015	-0.017	0.047

Triplet nu	Triplet caché	Choix de ligne	X-Wing	Y-Wing	Swordfish	Backtracking
-0.058	0.074	0.082	0.127	0.028	0.529	0.050

TABLE 2 – Poids des techniques

Afin d'affiner l'évaluation, j'ai ajouté pour chaque technique un coefficient constant β_i qui traduit l'utilisation ou non de la technique :

$$S \mapsto (n_1, \dots, n_p) \mapsto \sum_{i=1}^{p} n_i \alpha_i + \sum_{i=1}^{p} \beta_i \epsilon_i$$

où $\epsilon_i \in \{0, 1\} = \mathbb{1}_{n_i > 0}$.
Ces paramètres supplémentaires permettent d'affiner l'évaluation (Figure 8).

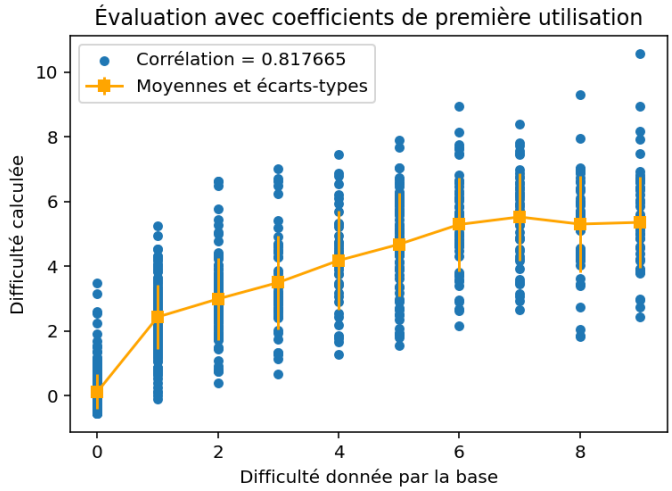


FIGURE 8 – Difficulté calculée en fonction de celle donnée par la base [5]

La corrélation est meilleure : on obtient ainsi une méthode assez fiable d'évaluation de la difficulté, et donc applicable à une large gamme de sudokus.

4 Lien avec le problème SAT

L'approche menée jusqu'à présent, bien que présentant des bons résultats, a l'inconvénient d'être limitée à l'évaluation d'un sudoku sans variantes. Cependant, elle peut servir de base pour

la construction d'une approche plus générique, basée sur le problème SAT. Cette approche pourrait alors s'étendre à des variantes du sudoku ou à d'autres jeux de logique.

L'objectif est donc d'établir une méthode d'évaluation de la difficulté d'un problème SAT en tant que problème logique pour humain.

4.1 Conversion

On considère une grille de sudoku : en représentant pour tous $i, j, k \in [\![1, n]\!]$ la présence ou non dans la case (i, j) du chiffre k par une variable booléenne $p_{i,j,k}$, les règles du sudoku peuvent être reformulées comme un ensemble de contraintes sur ces variables booléennes. La résolution d'une grille de sudoku peut alors se réduire à trouver un environnement propositionnel adapté à une instance du problème CNF-SAT.

Les contraintes sur les lignes, colonnes, blocs et cases se traduisent par des clauses "1 parmi 9" : exactement une des neuf variables concernées doivent être vraies pour satisfaire la clause.

Pour obtenir une formule de la logique propositionnelle sous forme 9-CNF, j'ai représenté l'exclusion mutuelle des variables par une exclusion deux-à-deux :

$$\left(\bigvee_{k=1}^{9} p_{i,j,k}\right) \wedge \left(\bigwedge_{k_1=1}^{9} \bigwedge_{k_2>k_1} \neg p_{i,j,k_1} \vee \neg p_{i,j,k_2}\right)$$

l'exemple ci-dessus impose la présence d'exactement un chiffre sur la case (i, j).

L'article [6] propose un encodage plus efficace en termes de nombre de clauses, mais on perd alors la correspondance "une variable = un chiffre dans une case".

4.2 Résolution

Le caractère relativement naturel de la conversion précédente permet notamment que l'application de l'algorithme de Quine se traduise immédiatement comme l'application des deux techniques basiques du sudokus, le singleton nu (un seul chiffre possible dans une case) et le singleton caché (une seule case possible sur une ligne/colonne/zone carrée pour un certain chiffre).

D'où l'idée d'établir un lien entre la résolution par Quine et la difficulté humaine du sudoku. J'ai donc implémenté, dans l'optique de reproduire le raisonnement humain, une variante de Quine : lorsque la formule traitée ne contient plus de clause à un élément, l'algorithme de Quine fait une disjonction de cas sur la valeur de vérité d'une variable, et s'appelle récursivement. Mon algorithme va traiter les deux cas de la disjonction en parallèle : ainsi, si les deux cas trouvent un résultat intermédiaire commun, alors ce résultat est nécessairement vrai : on peut alors arrêter la disjonction et admettre ce résultat.

En figure 9, par exemple, lors de la disjonction de cas sur la position du 6 dans la zone de gauche, chaque cas permet de placer le 6 vert au même endroit. On peut donc arrêter la disjonction (sans être arrivé à une absurdité) et valider le 6 vert. On a alors reproduit le raisonnement de la technique humaine de la "paire pointante".

6	6	1	2	6	6
4	2	5	3	9	6
7	8	9	1	4	5
		6			

FIGURE 9 – Après la disjonction sur les 6 bleus, les deux cas aboutissent à placer le 6 vert au même endroit. La disjonction s'arrête, le 6 vert est placé.

L'algorithme est détaillé en annexe 2.

Alors qu'un essai-erreur classique peut faire durer l'incertitude jusqu'à la fin de la grille, ce qui est peu motivant pour un joueur, ce procédé garantit de trouver l'erreur dès que possible, et permet de se contenter de déductions sur d'autres variables, impossibles avec l'essai-erreur.

D'autres méthodes [2] évaluent des sudokus à partir de l'instance de SAT associée, en examinant le comportement d'un algorithme de résolution (dans le cas de [2], les formules logiques sont transformées en problèmes continus et résolues dynamiquement)

4.3 Évaluation

Il s'agit maintenant d'estimer la difficulté de la grille à partir du déroulement de l'algorithme décrit précédemment.

J'ai envisagé trois critères :

— La profondeur de l'arbre des disjonctions
— Le nombre total de disjonctions
— Le nombre total d'éliminations de variables par Quine (Algo 1, ligne 13 / Algo 2, ligne 2)

Dans ce TIPE, je me suis limité à une difficulté de la forme :

$$d = \alpha(profondeur) + \beta \times n_d + \gamma \times n_Q$$

où n_d le nombre de disjonctions, et n_Q le nombre d'élimination de variables par Quine. Les paramètres β, γ et les valeurs de α seront déterminés pour coller à la base de données [5].

Cette méthode d'évaluation est choisie simple, car il y a aussi un autre facteur déterminant : l'heuristique de choix de variable pour la disjonction de cas. En effet, avoir une heuristique pertinente est aussi un facteur clé dans l'évaluation, puisque c'est un enjeu majeur de la résolution par Quine. J'ai donc comparé trois heuristiques :

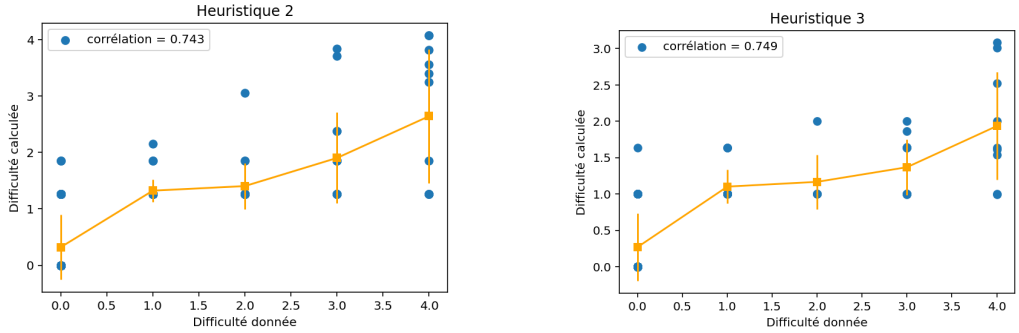
1. Choix aléatoire
2. Minimisation de la taille des clauses dont la variable fait partie
3. Maximisation de la taille des clauses dont la variable fait partie

L'heuristique 2 vise à avoir un effet immédiat, tandis que l'heuristique 3 cherche à maximiser son effet sur le long terme.

Les résultats (Table 3 et Figure 10) ci-après ont été obtenus sur 100 grilles pour l'entraînement et 100 pour le test.

heuristique	Corrélation	β	γ	$\alpha(0)$
1	Pas de résultats (trop lent)			
2	0,76	1,0	-0,03	-0,5
3	0,74	0,86	-0,003	-0,35

TABLE 3 – Résultats des heuristiques



(a) Heuristique 2 (b) Heuristique 3

FIGURE 10 – Résultats des heuristiques

Cette approche me donne donc une méthode pour l'évaluation de la difficulté d'un problème SAT, qui dans l'état actuel est une alternative un peu moins précise que la première méthode pour évaluer un sudoku, mais pourrait servir de base pour l'évaluation d'autres jeux de logique.

5 Conclusion

Ce TIPE a montré que les critères visuels ne constituent pas un moyen satisfaisant d'évaluer la difficulté, mais a mis en place deux méthodes d'évaluation pertinentes : la première reproduit le raisonnement humain directement sur un sudoku, tandis que la deuxième fait de même sur une instance du problème SAT associée. Cet alternative entraîne une légère perte de précision, mais pourrait permettre de généraliser cette méthode d'évaluation à d'autres jeux de logique.

Références

[1] Radek Pelánek. “Difficulty Rating of Sudoku Puzzles : An Overview and Evaluation”. In : (2014). doi : 10.48550/arXiv.1403.7373.

[2] Mária Ercsey-Ravasz & Zoltán Toroczkai. “The Chaos Within Sudoku, A Richter-type scale for Sudoku hardness”. In : *Sci Rep 2* (2012). doi : 10.48550/arXiv.1208.0370.

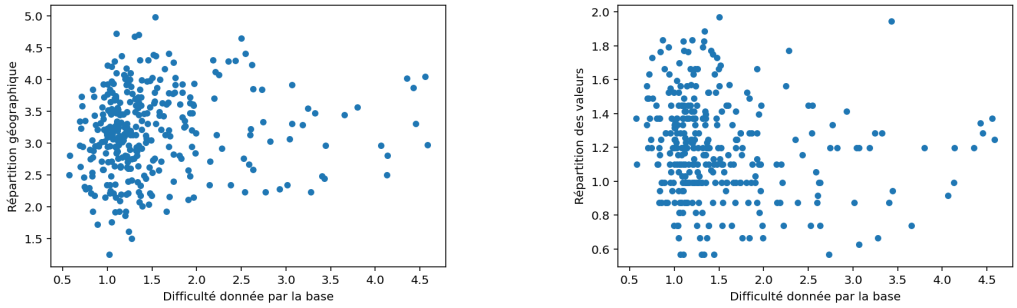
[3] Chungen Xu & Weng Xu. “The Model and Algorithm to Estimate the Difficulty Levels of Sudoku Puzzles”. In : *CCSE* (2009). doi : 10.5539/jmr.v1n2p43.

[4] Sheng-Wei Wang. “A Dataset of Sudoku Puzzles With Difficulty Metrics Experienced by Human Players”. In : *IEEE Access* 12 (2024), p. 104254-104262. doi : 10.1109/ACCESS.2024.3434632.

[5] url : https://huggingface.co/datasets/imone/sudoku-hard-v2.

[6] Will Klieber & Gihwon Kwon. “Efficient CNF Encoding for Selecting 1 from N Objec ts”. In : *cs.cmu.edu* (2007).

Annexe A Résultats des autres critères visuels



(a) Répartition géographique

(b) Répartition des valeurs

Figure 11 – Résultats des autres critères visuels

Les autres critères ne sont pas corrélés à la difficulté (corrélation de 0,1 pour les deux).

Annexe B Extraits du code

```
bool solve_notes(grid_t g, bool(**techniques)(grid_t g), int p) {
  /* Entrées : - g la grille à traiter */
  /* - techniques un tableau de pointeurs de fonctions, */
  /* qui sont les techniques de résolution (hors backtracking) */
  /* - p le nombre de techniques */
  /* Applique les techniques par ordre croissant de difficulté */

  int i = 0 ;
  while(i<p){
    if(techniques[i](g)){
      g->nb_techniques[i] ++ ;
      i = 0;
    }
    else{
      i++;
    }
    if(!est_ok(grid: g->grid)){
      return false;
    }
  }
```

Figure 12 – Fonction solve_notes

```c
struct grid_s {
    int** grid;
    bool*** notes;
    float* nb_techniques;
};
typedef struct grid_s* grid_t;

bool lastFreeCell(grid_t g);
int hiddenSingle(grid_t g);
bool nakedSingle(grid_t g);
bool nakedPair(grid_t g);
bool nakedTriple(grid_t g);
bool hiddenPair(grid_t g);
bool hiddenTriple(grid_t g);
bool pointingPair(grid_t g);
bool boxLineReduction(grid_t g);
bool x_wing(grid_t g);
bool y_wing(grid_t g);
bool swordfish(grid_t g);
```

FIGURE 13 – Signatures des techniques

Annexe C Algorithme de Quine modifié

Entrées : φ une CNF
Sorties : S_D l'ensemble des valeurs déduites

$v \leftarrow h(\varphi)$ /* Variable sur laquelle sera faite la disjonction. */
$S_V = \{(v, Vrai)\}$ /* Ensemble des valeurs déduites en supposant v vraie. */
$\varphi_V = \varphi[v \leftarrow Vrai]$ /* Formule déduite en supposant v vraie. */
$S_F = \{(v, Faux)\}$
$\varphi_F = \varphi[v \leftarrow Faux]$
tant que $S_V \cap S_F = \emptyset$ **faire**
  si *une clause de* φ_V *est vide* **alors**
    /* Alors φ_V est insatisfiable */
    retourner S_F /* Les assignations de variables trouvées sont donc correctes */
  fin
  sinon si *une clause de* φ_F *est vide* **alors**
    retourner S_V
  fin
  sinon si *un littéral* l *est seul dans une clause de* φ_V *(resp.* φ_F*)* **alors**
    Remplacer l par vrai et son complémentaire par faux dans φ_V (resp. φ_F)
  fin
  sinon
    $S_V \leftarrow Disjonction(\varphi_V)$
    ou
    $S_F \leftarrow Disjonction(\varphi_F)$
  fin
fin
retourner $S_V \cap S_F$

Algorithme 1 : Disjonction

Entrées : φ une CNF
Sorties : S un ensemble de couples (variable,valeur) qui satisfait φ

```
1 tant que φ est non vide faire
2 |  si un littéral l est seul dans une clause de φ alors
3 |  |  Remplacer l par vrai et son complémentaire par faux dans φ S ← V/F
4 |  sinon
5 |  |  S ← Disjontion(φ)
6 |  fin
7 fin
```

Algorithme 2 : Quine Modifié