Projet Recherche Opérationnelle :
Génération de Sudoku

Philippe LELEUX
Groupe F

15 janvier 2013

Résumé :

En Recherche Opérationnelle, nous allons étudier différentes manières manières :
- **de générer une grille de sudoku pleine : par recuit simulé et par permutations aléatoires avec améliorations d'un critère**
- **de vider la grille de sudoku au maximum de manière à garder une solution unique : grâce au recuit simulé et par backtracking**

Table des matières

Présentation générale

1. Introduction

On ne présente plus le principe du sudoku, jeu de réflexion où il faut remplir une grille avec les chiffres de 1 à 9 de manière à ce qu'ils apparaissent une seule fois par ligne, par colonne et par région. Ce jeu est en faite basé sur des mathématiques précises et peut être vu comme un problème de coloration de graphe par exemple (où chaque noeud est une case et est lié à toute case de la même colonne, ligne et région). Et plus généralement comme un problème de recherche opérationnelle.

Dans ce projet, je vais étudier deux des problèmes du sudoku :

- le premier, celui qui rend fous des milliers de personnes de par le monde, est la résolution de sudoku ou plus exactement la génération de grilles pleines. Comment obtenir une grille de sudoku valide à coup sûr et ce que l'on parte d'une grille vide ou non ?
- le second, comment vider une grille au maximum de manière à ce que celle-ci garde une solution unique ?

Le but réel de ce projet est d'étudier le fonctionnement du Recuit Simulé (méthode métaheuristique pour les calculs d'optimisation) que nous examinerons en détail dans la partie 1.

2. Spécifications et solutions

Pour générer des sudokus, la spécification est seulement que l'on foit obtenir des grilles valides tout simplement. J'ai fait le choix de pouvoir lancer mon programme de génération de sudokus en partant d'une grille vide, c'est-à-dire faire de la génération pure, ou d'une grille avec déjà certaines cases remplies (qui seront alors conservées), c'est-à-dire trouver une des solutions.

Dans ce cadre, la suite de ce rapport portera sur 3 méthodes de génération : la résolution de grille par backtracking, la génération de grille par permutations aléatoires avec amélioration d'un critère choisi et enfin le recuit simulé dont on va étudier l'efficacité pour cette application

Pour ce qui est de vider des grilles de sudoku, la spécification est seulement que celles-ci doivent garder une solution unique avec un minimum de valeurs renseignées et, biensûr, que l'on doit utiliser le recuit simulé.

Nous allons examiner deux méthodes pour résoudre ce problème : la force brute et le recuit simulé.

I Recuit Simulé

1. Principe

Le principe du recuit provient de la métallurgie et repose sur le principe que pour refroidir un métal, il faut de temps en temps lui apporter un peu de chaleur pour au final minimiser son énergie et qu'il soit le plus solide possible.

En pratique, pour un problème d'optimisation, cette méthode permet de ne pas trop vite converger vers un minimum local. En « franchissant un col », c'est-à-dire en passant par des grilles paraissant mauvaises par rapport à l'objectif on peut étendre le domaine de recherche pour trouver le minimum global.

2. Algorithme

L'état initial de l'algorithme doit être choisi dans l'espace des solutions, cet état dispose d'une énergie et d'une température initiale. Le but est de faire baisser au maximum l'énergie en baissant la température selon un principe choisi.

A chaque itération, on fait varier la solution de manière aléatoire. Si on fait baisser l'énergie, on accepte la modification, sinon on l'accepte si elle a une probabilité inférieure à $e^{-(delta/T)}$ où delta est la variation d'énergie avec l'itération précédente.

Il faut également faire varier la température et pour cela il y a deux politiques : soit on fait varier de manière constante à chaque itération, soit on attend un certain nombre d'itérations (constituant un palier) avant de faire baisser la température, en théorie la fin d'un palier constitue à un équilibre thermodynamique.

II Génération de sudokus

1. Les trois approches du problème

1.1 Backtracking

La première approche du problème est que celle du néophite, on prend un sudoku dans un magasine ou n'importe où et on essaye de le résoudre pour générer une grille pleine. Pour cela, les méthodes que l'on pourrait qualifier d' « humaines » (par exemple prendre une ligne de régions et essayer de remplir chiffre par chiffre) ne sont pas forcément complètes - elles ne donnent pas toujours la solution – et surtout sont fastidieuses à coder.

Le backtracking au contraire est relativement simple à coder et c'est une méthode complète – comment en serait-il autrement puisque l'algorithme parcourt l'ensemble des solutions possibles et s'arrête lorsqu'il en a une.

Pour cela il faut coder deux méthodes :

- la première vérifie pour chaque case si une seule valeur est possible en se

référant à la ligne, la colonne et la région. Si c'est le cas elle renvoie la solution, sinon elle appelle la deuxième
- celle-ci prend la première case vide et la remplie avec une solution possible avant d'appeler la première méthode. Si la première finit par renvoyer un sudoku null, elle essaye les autres valeurs possibles.

Le problème principal de cette fonction a été son fonctionnement récursif empéchant un debuggage rapide et efficace.

1.2 Permutations aléatoires avec amélioration de critère

Une deuxième approche se rapproche plus de la génération pure et simple de sudoku, c'est à dire créer une grille pleine en partant d'une grille totalement vide.

Pour eela, on commence par définir un critère qui guidera la génération : la somme sur toutes les cases de la somme des nombres d'occurences de chaque chiffre dans les zones atteignables par la case – ligne, colonne, région – sans compter la première occurrence (ouf !). Ce critère diminue lorsqu'on se rapproche d'une grille valide et vaut 0 si on en a une.

Ensuite, on permute deux cases choisies aléatoirement et on garde ce changement seulement si le critère est diminué.

Si cet algorithme peut paraître approprié également pour résoudre des grilles avec cases remplies au premier abord (en gardant les cases de départ fixes bien entendu), la pratique montre que ce n'est pas réellement le cas – on utilise alors la première version, résolution par backtracking. A mon avis cela est dû au fait qu'en raison des cases fixes, une fois arrivé dans un minimum local on ne peut en sortir, alors que pour arriver à LA solution, il faudrait s'en éloigner un tout petit peu. Tiens, cela ne vous rappelle-t-il pas quelque chose ?

Malheureusement c'est une approche que je n'ai pas eu le temps de tester y ayant pensé seulement peu avant la deadline.

1.3 Recuit Simulé

L'algorithme de recuit simulé que j'ai utilisé ici vient d'une publication de Renaud Sirdey, chercheur au Commissariat à l'énergie atomique :

http://sirdeyre.free.fr/Papiers_etc/2006_Sudokus_et_algorithmes_de_recuit_2.pdf

Dans cet algorithme, on utilise une itération par palier de 81 itérations avant de faire baisser la température selon une loi garantissant la convergence vers une loi stationnaire. Le critère d'arrêt est ue température minimale à ne pas dépasser.

2. Tests

2.1 Séries de tests

Nous allons tester l'efficacité des trois algorithmes dans divers problèmes, commençons donc par définir les buts des tests et les séries de tests qui en découlent.

But des tests :

- tester l'efficacité des méthodes face à des grilles vides
- idem avec des grilles non vides

Séries de tests :

- lancer chaque algorithme sur une grille vide
- lancer chaque algorithme sur une grille « facile »
- tester éventuellement sur une grille « très difficile »

2.2 Manuel d'utilisation : MainGenerer.java

Pour lancer la classe de tests de génération de sudokus : après avoir compilé, il faut taper : *java MainGenerer [fichier]*

Le paramètre *fichier* est facultatif et correspond au chemin relatif du fichier contenant la grille de sudoku initiale sous la forme :

700000400
020070080
003008009
000500300
060020090
001007006
000300900
030040060
009001005

Cette classe de test lance d'abord l'algorithme de permutation si la grille est vide ou l'algorithme de backtracking si elle est vide, puis le recuit simulé sur la même grille de départ.

2.3 Résultats des tests

voir annexe A.1.1.

2.4 Analyse des résultats

D'abord, après les tests on sait que la méthode des permutations aléatoires ne peut être utilisée dans tout les cas de résolution de sudokus comme je l'ai dit plus haut. Au contraire la résolution par backtracking est quasi-instantannée puisque le pire cas de résolution – une grille de niveau diabolique - a pris un peu moins de 4s.

Mais ce qui est le plus net après avoir effectué ces tests est que le recuit simulé n'est pas du tout applicable à ce genre de problème :

- c'est une méthode qui, si on veut un résultat satisfaisant, prend énormément de temps pour un problème de ce type (en dizaines de minutes là où les autres méthodes sont quasi-instantannées)

- cette méthode permet d'obtenir un résultat approché, se rapprochant d'un état stable par rapport à un critère mais pas du tout une solution exacte comme on la cherche. Selon la source de mon algorithme, il y a tout de même une probabilité de trouver la solution exacte. Même si j'ai pu voir l'algorithme s'en approcher, ça n'a jamais été le cas.

III Vider des sudokus

1. Les deux approches du problème

1.1 Brute Force

Une approche naïve du problème peut être d'examiner toutes les configurations de la grille à vider. Pour cela, il suffit d'incrémenter un compteur écrit en binaire de 0 jusqu'à 2^(nombre de cases de départ non vide)-1 : chaque case non vide correspond à un bit et sera enlevé si il est égal à 1.

Avec cette méthode on est ainsi sûr d'arriver à la solution optimale.

1.2 Recuit Simulé

L'algorithme de Recuit simulé que nous allons utiliser est librement codé à partir du principe du recuit simulé et en choisissant de faire évoluer la température à chaque itération de manière constante. Le critère d'arrêt choisi est là aussi une température minimale.

Dans cet algorithme, nous allons choisir à chaque étape une case :

- Si elle est pleine, on regarde si en la vidant on garde une solution unique et si c'est le cas on accepte la modification.
- Si elle est vide, si aléatoirement on trouve une probabilité inférieure à e^(-1/T) on fait cette modification – on appelle ça une dégradation.

La plus grosse difficulté de codage de cet algorithme est la fonction pour vérifier si la solution de la grille est unique. En réalité, la méthode nécessaire est exactement la même que la méthode de backtracking pour résoudre un sudoku sauf que cette fois-ci on va jusqu'au bout du parcours en comptant chaque solution possible.

2. Tests

2.1 Séries de tests

Soyons honnêtes, pour tester réellement l'algorithme de brute force codé, il faudrait des jours de calculs et même sûrement des semaines avec une grille complète.

Nous allons donc nous intéresser aux tests concernant le recuit simulé et les différences apportées par le choix de la température initiale, finale et du pas de décrémentation.

2.2 Manuel d'utilisation

Pour lancer la classe de tests pour vider un sudoku : après avoir compilé, il faut taper : *java MainProbleme [Ti] [Tf] [pas de décrémentation] [fichier]*

Tf et Ti correspondent aux températures finale et initiale que l'on souhaite, pas de décrémentation, le pas souhéaité. Le paramètre *fichier* est facultatif et correspond au chemin relatif du fichier contenant la grille de sudoku initiale.

Cette classe de test lance d'abord l'algorithme du recuit simulé avec les paramètres entrés puis lance l'algorithme de brute force sur la matrice trouvée.

2.3 Résultats des tests

voir annexe A.1.2.

2.4 Analyse des résultats

Il est assez difficile de définir exactement l'influence des paramètres sur la rapidité de l'algo et la qualité du résultat.

- On peut remarquer qu'il y a des paliers, pour donner un exemple : que ce soit en partant de la température 1 et en diminuant de 0.0002 à chaque itération, on obtient une énergie minimum de 22 que l'on aurait obtenue en s'arrêtant à une température de 0.15.
- Le réglage de la température initiale n'est pas ce qui importe réellement, tant qu'on la prend suffisamment éloignée de la température finale (éloignée ayant étant une notion relative au pas de décrémentation, 1000 fois le pas d'écart par exemple). Plus l'écart est grand plus le résultat sera précis.
- Lorsque l'on change le pas, on joue surtout sur la rapidité de l'algorithme (logique puisque le critère d'arrêt est la température finale) mais aussi sur la qualité du résultat, plus le pas est petit, plus l'algorithme sera lent mais précis..
- La température finale doit être suffisamment basse (0.01 par exemple). Cependant, il faut faire attention que lorsque la température devient assez basse, la probabilité de dégradation devient presque nulle et on resque coincé dans le minimum local courant.

Le meilleur résultat (et plus rapide) trouvé est avec :

Ti = 1
Tf = 0.15
pas = 0.0002

Conclusion

1. Génération de Sudokus

Comme on pouvait le penser, pour résoudre un sudoku, la méthode du backtracking est toute indiquée tandis qu'en partant d'une grille vide on pencherait plus pour la méthode des permutations aléatoires avec amélioration de critère.

Soyons clairs quand au recuit simulé, celui-ci n'est clairement pas intéressant pour ce cas.

2. Vider des Sudokus

En conclusion, pour obtenir une approximation du minimum rapidement, ce que l'on souhaite, le recuit simulé est tout à fait indiqué. La seule difficulté vient du choix des températures finale et initiale ainsi que du pas de décrémentation.

Ce qu'il faut retenir c'est qu'il faut un nombre d'itérations suffisant pour avoir le temps de converger vers un minimum local (et un bon de préférence) et que la température finale doit être assez petite pour y rester bloquer à partir d'un certain rang. Ainsi si la température est trop petite, la probabilité de dégrader est quasi-nulle donc cela ne sert plus à rien.

Annexes :

A Tests

A.1 Test Générer

A.1.a grille de niveau facile

La grille de départ est :
0 0 2 3 7 0 9 0 0
0 0 7 5 6 8 4 0 2
0 8 0 0 9 0 0 0 0
1 0 0 0 4 0 8 0 0
2 0 4 0 0 0 7 0 6
0 0 6 0 2 0 0 0 1
0 0 0 0 5 0 0 1 0
5 0 1 9 3 2 6 0 0
0 0 3 0 8 6 2 0 0

Résolution par backtracking
Résultat :
4 6 2 3 7 1 9 8 5
9 1 7 5 6 8 4 3 2
3 8 5 2 9 4 1 6 7
1 7 9 6 4 5 8 2 3
2 5 4 8 1 3 7 9 6
8 3 6 7 2 9 5 4 1
6 2 8 4 5 7 3 1 9
5 4 1 9 3 2 6 7 8
7 9 3 1 8 6 2 5 4

Durée d'exécution : 103ms
Résolution par recuit simulé avec avec amélioration du critère [cij] en cours...
9 6 2 3 7 0 9 4 5
7 5 7 5 6 8 4 5 2
5 8 8 4 9 8 5 0 8
1 4 9 8 4 3 8 0 3
2 1 4 3 3 7 7 0 6
2 1 6 3 2 0 6 9 1
8 7 4 9 5 8 4 1 2
5 3 1 9 3 2 6 5 0
2 7 3 3 8 6 2 0 2

0.9998824920097452

8 5 2 3 7 0 9 6 3
5 4 7 5 6 8 4 4 2
1 8 8 1 9 7 4 3 5
1 7 1 5 4 7 8 3 9
2 9 4 3 4 4 7 7 6
1 7 6 5 2 2 6 3 1
5 2 7 9 5 9 8 1 1
5 3 1 9 3 2 6 8 2
9 6 3 8 8 6 2 9 4

0.0029977204296318906

Le résultat final est :
7 7 2 3 7 0 9 6 4
1 3 7 5 6 8 4 7 2
3 8 6 2 9 4 7 9 5
1 7 8 4 4 3 8 5 1
2 1 4 1 1 5 7 8 6
2 3 6 5 2 9 2 7 1
8 8 9 9 5 4 9 1 4
5 3 1 9 3 2 6 6 4
3 9 3 8 8 6 2 5 6

Durée d'exécution : 3573484ms

A.1.b grille de niveau Très Difficile

La grille de départ est :
7 0 0 0 0 0 4 0 0
0 2 0 0 7 0 0 8 0
0 0 3 0 0 8 0 0 9
0 0 0 5 0 0 3 0 0
0 6 0 0 2 0 0 9 0
0 0 1 0 0 7 0 0 6
0 0 0 3 0 0 9 0 0
0 3 0 0 4 0 0 6 0
0 0 9 0 0 1 0 0 5

Résolution par backtracking
Résultat :
7 9 8 6 3 5 4 2 1
1 2 6 9 7 4 5 8 3
4 5 3 2 1 8 6 7 9
9 7 2 5 8 6 3 1 4
5 6 4 1 2 3 8 9 7

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

Durée d'exécution : 98ms
Génération par recuit simulé avec avec amélioration du critère [cij] en cours...
7 2 4 0 0 5 4 6 3
5 2 3 6 7 0 3 8 8
3 6 3 5 1 8 3 8 9
0 4 0 5 2 7 3 8 0
5 6 7 0 2 9 4 9 8
5 2 1 0 2 7 1 1 6
9 0 9 3 0 2 9 6 4
2 3 8 7 4 1 5 6 8
0 0 9 2 0 1 2 4 5

0.9998824920097452

7 2 5 5 4 2 4 8 3
1 2 8 7 7 4 6 8 4
2 1 3 3 7 8 1 1 9
3 4 5 5 8 4 3 7 8
1 6 9 0 2 1 7 9 8
5 2 1 6 5 7 2 7 6
2 4 5 3 6 9 9 3 2
6 3 6 7 4 5 9 6 6
3 8 9 9 1 1 8 4 5

0.005494997945049297

Le résultat final est :
7 2 4 2 6 2 4 3 3
1 2 3 1 7 1 9 8 9
5 7 3 4 7 8 9 7 9
4 8 1 5 7 8 3 5 9
6 6 2 0 2 5 5 9 8
1 4 1 4 6 7 6 7 6
5 8 2 3 2 2 9 1 8
9 3 5 4 4 3 8 6 5
8 4 9 3 6 1 6 1 5

Durée d'exécution : 632417ms

A.1.c grille vide

La grille de départ est :
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0

Génération par permutations aléatoires avec amélioration du critère [nombre de surnuméraires] en cours...

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

Durée d'exécution : 582ms
Génération par recuit simulé avec avec amélioration du critère [cij] en cours...
0 2 7 1 8 9 0 0 0
8 2 5 7 0 1 8 0 3
0 0 0 3 4 9 0 0 6
6 9 0 0 1 0 1 5 1
7 5 8 6 5 0 7 0 0
7 7 9 3 7 6 0 4 0
0 6 0 9 0 4 6 8 6
0 5 7 0 2 9 0 3 4
1 8 0 2 0 1 0 1 5

0.9998824920097452

1 1 9 4 4 4 1 8 9
5 2 7 8 4 1 6 0 2
9 9 7 2 6 1 9 8 8
6 5 5 3 4 6 7 3 8
6 4 6 7 2 8 2 7 7

3 2 8 2 8 4 7 7 1
6 1 5 3 9 3 5 4 9
7 3 9 6 1 8 5 2 3
2 9 4 3 0 5 3 6 5

0.005494997945049297

Le résultat final est :
4 4 6 7 7 8 9 5 9
3 5 4 4 8 6 1 0 5
2 1 1 8 6 4 7 7 2
8 6 3 7 5 7 5 1 3
7 3 2 6 9 1 9 2 2
6 1 8 5 6 2 6 5 3
8 2 4 8 4 8 9 9 9
7 9 7 3 3 1 5 3 2
6 1 4 2 0 1 3 5 8

Durée d'exécution : 568550ms

A.1.d grille de nombre minimum 17

Ti = 1, Tf = 0.15, pas = 0.0002

La grille de départ est :
0 0 0 0 0 0 0 1 0
4 0 0 0 0 0 0 0 0
0 2 0 0 0 0 0 0 0
0 0 0 0 5 0 4 0 7
0 0 8 0 0 0 3 0 0
0 0 1 0 9 0 0 0 0
3 0 0 4 0 0 2 0 0
0 5 0 1 0 0 0 0 0
0 0 0 8 0 6 0 0 0

Résolution par backtracking
Résultat :
6 9 3 7 8 4 5 1 2
4 8 7 5 1 2 9 3 6
1 2 5 9 6 3 8 7 4
9 3 2 6 5 1 4 8 7
5 6 8 2 4 7 3 9 1
7 4 1 3 9 8 6 2 5
3 1 9 4 7 5 2 6 8
8 5 6 1 2 9 7 4 3
2 7 4 8 3 6 1 5 9

Durée d'exécution : 3020ms
Génération par recuit simulé avec avec amélioration du critère [cij] en cours...
4 5 6 1 2 0 0 1 7
4 6 6 9 5 5 7 0 5
0 2 3 7 7 1 7 0 3
2 7 4 4 5 0 4 6 7
0 0 8 0 0 6 3 1 2
7 0 1 4 9 8 2 0 3
3 8 0 4 0 0 2 5 5
0 5 8 1 1 2 6 8 3
6 9 0 8 7 6 9 2 1

0.9998824920097452

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

0.005494997945049297

Le résultat final est :
2 7 1 9 7 8 1 1 8
4 4 7 7 6 5 6 3 7
3 2 2 9 4 6 7 5 9
4 7 2 2 5 5 4 1 7
1 8 8 6 6 4 3 8 9
5 9 1 4 9 4 7 2 6
3 1 8 4 3 9 2 3 2
5 5 8 1 8 9 5 2 5
3 3 3 8 1 6 9 6 6

Durée d'exécution : 610111ms

A.2 Test Vider Sudoku

A.2.a grille de niveau facile 1

Ti = 5, Tf = 0.05, pas = 0.0002

T =5.0
E =81

9 4 1 2 5 7 3 6 8
5 0 6 9 3 1 7 4 2
7 2 3 8 6 4 9 5 1
8 7 9 6 1 5 2 3 4
2 6 4 3 8 9 5 1 7
3 1 5 7 4 2 6 8 9
1 9 8 5 2 3 4 7 6
6 5 7 4 9 8 1 2 3
4 3 2 1 7 6 8 9 5

amelioration critere:
i = 1,j = 1
E = 80

9 4 0 0 0 7 3 6 8
5 8 6 9 0 1 7 0 0
7 2 3 8 0 0 9 0 1
8 7 0 0 0 5 2 3 0
2 6 0 0 0 0 0 0 0
0 1 0 0 0 0 0 8 9
0 9 0 5 2 0 4 0 0
0 5 7 4 9 0 1 2 0
0 3 0 1 0 6 8 0 0

amelioration critere:
i = 4,j = 4
E = 42

Grille vidée :
0 0 1 0 5 7 0 0 0
0 0 0 0 0 0 7 4 0
0 0 0 8 6 0 0 0 0
0 7 0 6 0 0 0 0 0
0 0 0 0 8 9 0 0 0
3 1 0 0 0 0 0 0 0
0 9 0 0 0 3 4 7 0
6 0 0 0 0 0 1 0 3

4 0 2 0 7 0 0 0 0

nombre de dégradations effectuées : 944
E = 23, T = 0.049999999998297816

La meilleure solution trouvée est :
0 0 1 0 5 7 0 0 0
0 0 0 0 0 0 7 4 0
0 0 0 8 6 0 0 0 0
0 7 0 6 0 0 0 0 0
0 0 0 3 8 9 0 0 0
3 1 0 0 0 0 0 0 0
1 9 0 0 0 3 4 7 0
6 0 0 0 0 0 1 0 3
0 0 2 0 0 0 0 0 0
avec E = 23

A.2.b grille de niveau facile 2

Ti = 1, Tf = 0.0004, pas = 0.00001
T =1.0
E =81

7 5 6 4 2 8 1 3 9
4 8 3 5 1 9 6 2 7
1 9 2 3 6 7 5 4 8
2 6 1 8 7 3 9 5 4
5 4 8 1 9 6 2 7 3
9 3 7 2 4 5 8 1 6
6 2 0 7 8 4 3 9 1
8 1 4 9 3 2 7 6 5
3 7 9 6 5 1 4 8 2

amelioration critere:
i = 6,j = 2
E = 80

T =0.4945599999994945
E =43

7 5 6 0 0 0 1 3 0
0 0 0 5 0 9 6 0 7
0 0 2 3 6 7 0 4 8
2 6 0 8 7 0 0 5 0
5 0 0 0 0 6 0 7 0

0 0 0 0 4 0 0 1 6
6 0 5 0 0 4 0 9 1
8 1 4 9 0 2 7 0 0
3 7 9 0 5 0 4 0 0

amelioration critere:
i = 2,j = 1
E = 42

Grille vidée : 7 0 0 4 0 0 0 0 0
0 8 0 0 0 0 6 2 0
1 0 0 0 6 7 0 0 0
0 6 0 8 0 0 0 0 0
5 0 0 0 0 0 0 0 3
0 0 0 0 0 5 8 0 0
0 0 0 0 0 0 0 9 1
0 0 0 0 0 0 0 0 5
3 7 9 6 0 0 0 0 2

nombre de dégradations effectuées : 11065
E = 22, T = 9.999999928148669E-5

La meilleure solution trouvée est :
0 0 0 4 0 0 0 0 0
4 8 0 0 0 0 6 2 0
1 0 0 0 6 7 0 0 0
0 6 0 8 0 3 0 0 0
5 0 0 0 0 0 0 0 3
0 0 0 0 0 5 8 0 0
0 0 0 0 0 0 0 9 1
0 0 0 0 0 0 0 0 5
0 7 9 6 0 0 0 0 2
avec E = 22

A.2.c grille de niveau facile 3

Ti = 5, Tf = 0.0001, pas = 0.0001
T =5.0
E =81

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

3 9 2 7 8 0 4 1 5
9 2 1 8 3 7 5 6 4
6 5 3 9 2 4 1 8 7
8 4 7 1 6 5 9 2 3

amelioration critere:
i = 5,j = 5
E = 80

T =1.4850999999981669
E =41

degrade si prob:
i = 7 ,j = 7
aleatoire = 0.09394240949722439

T =1.484999999998167
E =42

degrade si prob:
i = 2 ,j = 7
aleatoire = 0.008593572560548268

T =1.484899999998167
E =43

0 0 9 3 1 2 0 5 0
1 0 0 5 7 0 2 4 9
0 0 0 6 4 0 8 3 0
5 1 8 0 0 0 6 7 2
7 0 4 2 5 0 0 0 0
3 0 0 0 0 6 4 1 0
9 2 0 0 0 7 5 6 0
0 5 0 0 0 0 1 8 0
8 0 7 1 0 0 9 2 3

amelioration critere:
i = 8,j = 0
E = 43

Grille vidée :
0 8 0 0 0 0 0 5 0
0 3 6 0 0 0 2 4 9
0 0 0 0 4 9 0 0 0
0 0 0 0 9 3 0 0 0
0 0 4 2 0 0 0 0 0

0 0 0 0 0 0 0 1 5
0 2 0 8 0 0 0 0 4
0 5 0 0 0 0 0 0 0
8 0 0 1 0 0 0 2 3

nombre de dégradations effectuées : 2192
E = 23, T = 9.999999831411618E-5

La meilleure solution trouvée est :
0 8 9 3 0 0 0 0 0
0 0 0 0 0 0 2 4 0
0 0 5 0 4 0 0 0 1
0 0 0 0 0 3 0 0 0
0 6 4 0 0 0 0 9 0
3 0 0 0 8 0 0 0 0
0 2 0 0 0 0 0 6 4
0 5 0 9 0 0 0 0 0
8 0 0 1 0 5 0 0 3
avec E = 23

A.2.d grille de niveau facile 4

Ti = 1, Tf = 0.00002, ps = 0.0001
T =1.0
E =81

4 8 3 1 7 2 9 6 5
1 7 9 5 6 8 4 3 2
5 2 6 9 4 0 8 7 1
6 9 7 3 2 5 1 8 4
8 1 4 7 9 6 2 5 3
3 5 2 8 1 4 6 9 7
2 3 5 4 8 9 7 1 6
7 6 8 2 5 1 3 4 9
9 4 1 6 3 7 5 2 8

amelioration critere:
i = 2,j = 5
E = 80

T =0.43027999999943023
E =45

degrade si prob:
i = 2 ,j = 7

aleatoire = 0.05235356860348876

T =0.4302599999994302
E =46

0 8 0 1 7 2 0 6 0
0 0 9 0 6 8 4 0 0
5 2 6 0 0 3 0 7 1
6 0 7 3 0 0 1 0 4
8 0 4 7 9 6 2 5 3
3 5 2 0 1 0 6 0 7
2 3 0 0 0 0 0 0 6
7 0 0 0 5 1 0 4 9
9 0 0 0 0 7 0 0 8

amelioration critere:
i = 1,j = 7
E = 45

Grille vidée :
0 8 3 1 0 2 0 0 0
0 0 9 0 6 0 0 0 0
0 0 0 0 0 0 0 0 0
6 0 0 3 0 0 1 0 0
0 0 0 0 0 0 2 0 0
0 0 0 8 1 0 6 9 0
2 0 5 0 0 9 0 0 6
7 0 0 0 0 0 3 0 0
0 4 0 0 0 7 0 0 0

nombre de dégradations effectuées : 11056
E = 22, T = 9.999999928148669E-5

La meilleure solution trouvée est :
0 8 3 0 7 0 0 0 5
0 7 0 0 6 0 0 0 0
0 0 0 0 0 0 0 0 0
6 0 0 3 0 0 1 0 0
0 0 0 0 0 0 2 0 0
0 0 0 0 0 4 0 9 7
2 0 5 0 0 9 7 0 0
0 0 0 2 0 0 3 0 0
9 0 1 6 0 0 0 0 0
avec E = 22

A.2.e grille de niveau facile 5

Ti = 5000, Tf = 1, pas = 0.02
T =5000.0
E =81

3 7 9 2 4 8 1 6 5
8 6 1 3 9 5 7 2 4
4 5 2 7 6 1 9 8 3
5 8 4 6 1 7 2 3 9
9 1 7 5 3 2 6 4 8
6 2 3 9 8 4 5 7 1
1 3 6 4 2 9 8 5 7
2 9 5 8 7 3 0 1 6
7 4 8 1 5 6 3 9 2

amelioration critere:
i = 7,j = 6
E = 80

T =2654.4000000021333
E =39

0 7 9 0 0 8 1 6 5
8 6 1 0 0 0 7 0 0
0 5 2 0 6 0 9 0 3
5 8 0 0 1 0 0 3 0
0 0 7 5 3 2 0 4 0
0 0 0 0 8 0 0 0 1
0 0 6 4 0 0 0 5 7
0 9 0 8 7 0 0 1 6
0 4 0 0 5 6 0 0 0

amelioration critere:
i = 4,j = 1
E = 38

Grille vidée :
3 7 0 2 0 8 0 6 0
8 0 0 3 0 5 0 0 0
0 5 2 7 6 1 0 0 3
5 8 4 0 1 7 2 3 9
0 1 0 0 3 2 0 4 8
0 2 3 9 0 0 5 0 0
1 0 0 4 2 0 8 5 7
0 9 0 8 0 0 0 1 6

7 4 8 1 0 0 0 9 0

nombre de dégradations effectuées : 0
E = 46, T = 0.8000000023561262

La meilleure solution trouvée est :
0 0 0 2 0 8 0 0 5
0 0 1 3 0 0 0 0 4
0 0 2 0 0 1 9 8 3
5 8 4 6 0 0 2 0 0
0 1 0 0 3 0 0 0 8
6 0 0 9 0 0 0 0 1
0 0 0 0 0 9 0 0 0
2 9 0 0 0 0 4 0 0
7 0 0 0 0 6 0 9 0
avec E = 29

A.2.f grille de nombre minimum 17

Ti = 1, Tf = 0.15, pas = 0.0001
La grille de départ est :
6 9 3 7 8 4 5 1 2
4 8 7 5 1 2 9 3 6
1 2 5 9 6 3 8 7 4
9 3 2 6 5 1 4 8 7
5 6 8 2 4 7 3 9 1
7 4 1 3 9 8 6 2 5
3 1 9 4 7 5 2 6 8
8 5 6 1 2 9 7 4 3
2 7 4 8 3 6 1 5 9

Vidage de la grille par recuit simulé en cours...
T =1.0
E =81

6 9 3 7 8 4 5 1 2
4 8 7 5 1 2 9 3 6
1 2 5 9 6 3 8 7 4
9 3 2 6 0 1 4 8 7
5 6 8 2 4 7 3 9 1
7 4 1 3 9 8 6 2 5
3 1 9 4 7 5 2 6 8
8 5 6 1 2 9 7 4 3
2 7 4 8 3 6 1 5 9

amelioration critere:
i = 3,j = 4
E = 80

T =0.6054000000000435
E =45

0 9 0 0 0 0 0 1 0
4 8 0 5 1 2 0 3 6
1 0 5 9 6 3 8 0 0
9 3 0 6 5 1 4 0 7
0 6 8 0 4 0 3 9 0
7 0 0 0 0 0 0 2 5
3 1 9 0 7 0 0 0 0
0 5 0 1 2 9 0 0 0
0 0 4 8 3 6 1 0 9

amelioration critere:
i = 5,j = 2
E = 44

T =0.6052000000000435
E =44

degrade si prob:
i = 6 ,j = 3
aleatoire = 0.06693189662636582

T =0.6050000000000435
E =45

Grille vidée :
6 0 3 0 0 0 5 0 0
0 0 7 5 1 0 0 0 0
0 0 5 9 0 0 0 0 4
9 0 0 0 0 0 0 0 0
5 0 0 0 0 7 3 0 0
0 4 0 0 0 8 0 2 0
3 1 0 4 0 0 0 6 0
0 0 0 0 0 0 7 0 0
0 0 0 0 0 6 0 0 0

nombre de dégradations effectuées : 579
E = 22, T = 0.14980000000007973

La meilleure solution trouvée est :

6 0 3 0 0 0 5 0 0
0 0 7 5 1 0 0 0 0
0 0 5 9 0 0 0 0 4
9 0 0 0 0 0 0 0 0
5 0 0 0 0 7 3 0 0
0 4 0 0 0 8 0 2 0
3 1 0 4 0 0 0 0 0
0 0 0 0 0 0 7 4 0
0 0 0 0 0 6 0 0 0
avec E = 22
Durée d'exécution : 5490ms