Sudoku_resolus_par_algorithme_genetique

May 21, 2019

1 Table of Contents

2 Bonus d'implémentation - Agrégation Option Informatique

2.1 Préparation à l'agrégation - ENS de Rennes, 2017-18

- *Date* : 20 février 2018
- *Auteur* : Lilian Besson
- *Texte*: Annale 2006, "Sudoku"

2.2 À propos de ce document

- Ceci est une *proposition* d'un bonus d'implémentation, pour un texte d'annale de l'agrégation de mathématiques, option informatique.
- Ce document est un notebook Jupyter, et est open-source sous Licence MIT sur GitHub, comme les autres solutions de textes de modélisation que j'ai écrite cette année.
- L'implémentation sera faite en OCaml, version 4+ :

```
In [1]: Sys.command "ocaml -version";;

The OCaml toplevel, version 4.04.2


Out[1]: - : int = 0
```

2.3 Vérifier qu'un Su Doku est correct

- On va implémenter rapidement la vérification des contraintes d'un Su Doku.
- En plus de répondre binairement si le Su Doku est correct, en comptant le nombre de contraintes satisfaites, on a une mesure numérique qu'on doit faire augmenter pour résoudre le Su Doku.

2.3.1 Fonctions utilitaires

```
In [14]: let ligne p t i = Array.init p (fun j -> t.(i).(j)) ;;
         (* t.(i) marche aussi bien ! *)

         let colonne p t j = Array.init p (fun i -> t.(i).(j)) ;;

Out[14]: val ligne : int -> 'a array array -> int -> 'a array = <fun>


Out[14]: val colonne : int -> 'a array array -> int -> 'a array = <fun>


In [15]: let tousVrai = Array.for_all (fun x -> x);;

Out[15]: val tousVrai : bool array -> bool = <fun>


In [16]: let estNp p tab =
             if tousVrai (Array.map (fun x -> (1 <= x) && (x <= p)) tab) then begin
                 let estLa = Array.make p false in
                 for i = 0 to p - 1 do
                     estLa.(tab.(i) - 1) <- true
                 done;
                 tousVrai estLa
             end
             else
                 false
         ;;

Out[16]: val estNp : int -> int array -> bool = <fun>
```

```
In [24]: let sum_array = Array.fold_left (+) 0;;

Out[24]: val sum_array : int array -> int = <fun>


In [25]: let estNp_compte p tab =
             if tousVrai (Array.map (fun x -> (1 <= x) && (x <= p)) tab) then begin
                 let estLa = Array.make p 0 in
                 for i = 0 to p - 1 do
                     estLa.(tab.(i) - 1) <- 1
                 done;
                 sum_array estLa
             end
             else
                 0
         ;;

Out[25]: val estNp_compte : int -> int array -> int = <fun>


In [26]: let racine_carree i = int_of_float (sqrt (float_of_int i));;

Out[26]: val racine_carree : int -> int = <fun>
```

2.3.2 Contraintes sur les lignes et les colonnes

```
In [18]: let contraintes_lignes p t =
             tousVrai (Array.init p (fun i ->
                     estNp p (ligne p t i)
                 ))
         ;;

Out[18]: val contraintes_lignes : int -> int array array -> bool = <fun>


In [27]: let contraintes_lignes_compte p t =
             sum_array (Array.init p (fun i ->
                     estNp_compte p (ligne p t i)
                 ))
         ;;

Out[27]: val contraintes_lignes_compte : int -> int array array -> int = <fun>


In [19]: let contraintes_colonnes p t =
             tousVrai (Array.init p (fun j ->
                     estNp p (colonne p t j)
                 ))
         ;;
```

```
Out[19]: val contraintes_colonnes : int -> int array array -> bool = <fun>
```

```
In [28]: let contraintes_colonnes_compte p t =
             sum_array (Array.init p (fun j ->
                     estNp_compte p (colonne p t j)
                 ))
         ;;
```

```
Out[28]: val contraintes_colonnes_compte : int -> int array array -> int = <fun>
```

```
In [20]: let carre_latin p t =
             (contraintes_lignes p t) && (contraintes_colonnes p t)
         ;;
```

```
Out[20]: val carre_latin : int -> int array array -> bool = <fun>
```

```
In [29]: let carre_latin_compte p t =
             (contraintes_lignes_compte p t) + (contraintes_colonnes_compte p t)
         ;;
```

```
Out[29]: val carre_latin_compte : int -> int array array -> int = <fun>
```

2.3.3 Contraintes sur les carrés

```
In [21]: let petit_carre p n t i j =
             Array.init p (fun k ->
                 t.(n*i + (k / n)).(n*j + (k mod n))
             )
         ;;
```

```
Out[21]: val petit_carre : int -> int -> 'a array array -> int -> int -> 'a array =
           <fun>
```

```
In [22]: let petits_carres_sont_latins p t =
             let n = racine_carree p in
             (* Par flemme, on créé le tableau entier, qu'on vérifie après *)
             let contraintes_petits_carres =
                 Array.init p (fun i ->
                     estNp p (petit_carre p n t (i / n) (i mod n) )
                 )
             in
             (* Mais on peut mieux faire, avec une boucle while par exemple, on sort dès qu'un
             tousVrai contraintes_petits_carres
         ;;
```

```
Out[22]: val petits_carres_sont_latins : int -> int array array -> bool = <fun>
```

```
In [31]: let petits_carres_sont_latins_compte p t =
             let n = racine_carree p in
             let contraintes_petits_carres =
                 Array.init p (fun i ->
                     estNp_compte p (petit_carre p n t (i / n) (i mod n) )
                 )
             in
             sum_array contraintes_petits_carres
         ;;

Out[31]: val petits_carres_sont_latins_compte : int -> int array array -> int = <fun>
```

2.3.4 Contraintes globales

```
In [32]: let est_resolu p t =
             (carre_latin p t) && (petits_carres_sont_latins p t)
         ;;

Out[32]: val est_resolu : int -> int array array -> bool = <fun>
```

```
In [50]: let nb_contraintes_resolues p t =
             (carre_latin_compte p t) + (petits_carres_sont_latins_compte p t)
         ;;

         let score = nb_contraintes_resolues;;

Out[50]: val nb_contraintes_resolues : int -> int array array -> int = <fun>


Out[50]: val score : int -> int array array -> int = <fun>
```

2.4 Exemples de “score”

2.4.1 Un exemple de Su Doku de taille 9×9 au compte juste

Avec $p = n^2 = 9$, on reprend l'exemple du texte :

Ça va être long un peu à écrire, mais au moins on vérifiera notre fonction sur un vrai exemple.

```
In [49]: let p = 9 ;;
         let t = [|
             [| 1; 2; 7; 4; 6; 3; 9; 8; 5 |];
             [| 3; 4; 9; 8; 7; 5; 2; 6; 1 |];
```

```
            [| 5; 8; 6; 2; 9; 1; 4; 3; 7 |];
            [| 7; 6; 5; 9; 4; 2; 3; 1; 8 |];
            [| 8; 3; 4; 7; 1; 6; 5; 2; 9 |];
            [| 9; 1; 2; 5; 3; 8; 7; 4; 6 |];
            [| 2; 7; 8; 6; 5; 4; 1; 9; 3 |];
            [| 4; 5; 3; 1; 8; 9; 6; 7; 2 |];
            [| 6; 9; 1; 3; 2; 7; 8; 5; 4 |];
        |];
```

```
Out[49]: val p : int = 9
```

```
Out[49]: val t : int array array =
           [|[|1; 2; 7; 4; 6; 3; 9; 8; 5|]; [|3; 4; 9; 8; 7; 5; 2; 6; 1|];
             [|5; 8; 6; 2; 9; 1; 4; 3; 7|]; [|7; 6; 5; 9; 4; 2; 3; 1; 8|];
             [|8; 3; 4; 7; 1; 6; 5; 2; 9|]; [|9; 1; 2; 5; 3; 8; 7; 4; 6|];
             [|2; 7; 8; 6; 5; 4; 1; 9; 3|]; [|4; 5; 3; 1; 8; 9; 6; 7; 2|];
             [|6; 9; 1; 3; 2; 7; 8; 5; 4|]|]
```

```
In [55]: let score_max = 3 * 9 * 9;;
```

```
Out[55]: val score_max : int = 243
```

```
In [56]: score p t;;
```

```
Out[56]: - : int = 243
```

2.4.2 Un exemple de Su Doku au *comte* faux

Avec $p = n^2 = 9$, en modifiant seulement une case du tableau T précédent.

```
In [57]: let p = 9 ;;
         let t = [|
             [| 1; 2; 7; 4; 6; 3; 9; 8; 5 |];
             [| 3; 4; 9; 8; 7; 5; 2; 6; 1 |];
             [| 5; 8; 6; 2; 9; 1; 4; 3; 7 |];
             [| 7; 6; 5; 9; 4; 2; 3; 1; 8 |];
             [| 8; 2; 4; 7; 1; 6; 5; 2; 9 |]; (* Ligne non valable, 2 est là deux fois !*)
             [| 9; 1; 2; 5; 3; 8; 7; 4; 6 |];
             [| 2; 7; 8; 6; 5; 4; 1; 9; 3 |];
             [| 4; 5; 3; 1; 8; 9; 6; 7; 2 |];
             [| 6; 9; 1; 3; 2; 7; 8; 5; 4 |];
         |];
```

```
Out[57]: val p : int = 9
```

```
Out[57]: val t : int array array =
           [|[|1; 2; 7; 4; 6; 3; 9; 8; 5|]; [|3; 4; 9; 8; 7; 5; 2; 6; 1|];
             [|5; 8; 6; 2; 9; 1; 4; 3; 7|]; [|7; 6; 5; 9; 4; 2; 3; 1; 8|];
             [|8; 2; 4; 7; 1; 6; 5; 2; 9|]; [|9; 1; 2; 5; 3; 8; 7; 4; 6|];
             [|2; 7; 8; 6; 5; 4; 1; 9; 3|]; [|4; 5; 3; 1; 8; 9; 6; 7; 2|];
             [|6; 9; 1; 3; 2; 7; 8; 5; 4|]|]
```

```
In [60]: score p t;;

Out[60]: - : int = 240
```

On voit que 3 contraintes ne sont pas vérifiées.

2.4.3 Un exemple de comte... Dooku ?

Nan, je déconne. ... Bien-sûr, évitez les blagues pourries le jour de l'oral ! Mais une bonne blague peut être bien reçue...

2.5 Algorithme génétique

- Maintenant qu'on peut mesurer à quelle "distance" on est d'un Su Doku complètement résolu (on appelera ça un *score* - ou "fitness" en anglais), on peut essayer de compléter aléatoirement la grille par algorithme génétique.
- Rappel de l'idée :

  1. on commence avec la grille, on fait N (= 100) fois un petit changement pour avoir une population initiale de solutions,
  2. pour G générations, on garde N1 (= 45) individus, inchangés, on efface N2 (= 10) individus qu'on retirera aléatoirement à partir de la grille initiale, et on fait "muter" (N3 = 45) individus, en ajoutant un seul chiffre aléatoirement.
  3. En pratique, pour améliorer la convergence de l'algorithme, on va effacer les N2 individus ayant le plus petit score, et faire muter la moitié des N-N2 (= 90) individus restant.

2.5.1 Configuration

```
In [34]: let population = 100
         and mutes = 45
         and effaces = 10;;
         let conserves = population - mutes - effaces;;
```

```
Out[34]: val population : int = 100
         val mutes : int = 45
         val effaces : int = 10


Out[34]: val conserves : int = 45


In [36]: Random.self_init();;

Out[36]: - : unit = ()


In [62]: Random.int;;

Out[62]: - : int -> int = <fun>


In [63]: let choix_sans_remise tab k =
             let n = Array.length tab in
             assert 0 <= k <= n;
             let reponse = Array.make k (-1) in
             for i = 0 to k-1 do

             done;
             reponse
         ;;


        File "[63]", line 1, characters 0-13:
    Error: Unbound value Random.choice
       1: Random.choice;;
```

2.5.2 Génération d'individus, mutations

```
In [41]: let copie n t =
             Array.init n (fun i -> (Array.copy t.(i)))
         ;;

Out[41]: val copie : int -> 'a array array -> 'a array array = <fun>


In [ ]: let individu_initial n grille_initiale =
            Array.init
        ;;

In [61]: let mutation n grille_initiale individu =
             XXX
         ;;
```

```
    File "[61]", line 2, characters 4-7:
Error: Unbound constructor XXX
   1: let mutation n grille_initiale individu =
   2:     XXX
   3: ;;
```

2.5.3 Algorithme génétique

```
In [ ]: let algorithme_genetique nb1 nb2 nb3 ii m generation =
            let n = nb1 + nb2 + nb3 in
            let population = List.init (fun i -> ii ()) in
```

```
In [ ]: let resolution_genetique_sudoku n grille_initiale =
            let ii = fun () -> individu_initial n grille_initiale in
            let m = fun individu -> mutation n grille_initiale individu in
```

```
In [ ]:
```

2.5.4 Essai

2.6 Conclusion

Voilà pour une proposition de bonus d'implémentation.

Bien-sûr, ce petit notebook ne se prétend pas être une solution optimale, ni exhaustive.