Resolución de sudokus mediante Algoritmos Genéticos

Alejandro Del Hierro Diez

14 de diciembre de 2018

UNIVERSIDAD DE VALLADOLID
ALGORITMOS Y COMPUTACIÓN
TRABAJO INDIVIDUAL
Curso 2018-2019

Índice

1. Introducción.

En este informe se propondrá un algoritmo genético que resuelva sudokus y se explicarán sus diferentes partes, así como un análisis del coste de dicho algoritmo en la búsqueda de una solución.

1.1. Breve introducción a los Algoritmos Genéticos

Este tipo de algoritmos simulan modelos de selección natural propuestos por Charles Darwin. En la resolución de problemas como el que se trata aquí, esto se traduce en generar una población de individuos (o posibles soluciones) de los que se escogerán los mas aptos, reproducirlos en función de su nivel de adaptación y añadir posibles mutaciones. Esto se repetirá hasta que se alcance la solución o hasta que se alcance un numero determinado de generaciones.

1.2. Aplicación a los sudokus

Un sudoku es un puzzle, cuya forma clásica consta de una rejilla de 9x9 casillas en las que hay que colocar numeros del 1 al 9 con una serie de restricciones:

- Filas: No será valido un sudoku que tenga en una fila números repetidos.
- Columnas: De la misma manera, no será correcto un sudoku con números repetidos en las columnas.
- Bloques: No podrá haber números repetidos en un bloque. Los bloques vienen dados por las rejillas de 3x3 casillas resultantes de dividir el sudoku original en 9 bloques diferentes.

Los sudokus se codificarán en matrices de dos dimensiones. Inicialmente se creará una población de sudokus a partir de una plantilla dada en un archivo de texto, rellenando aleatoriamente las casillas vacías. A partir de aquí se iniciará el proceso iterativo explicado anteriormente.

2. Algoritmo.

A continuación se explicara en detalle el funcionamiento del algoritmo utilizado, así como el funcionamiento de todos los componentes del algoritmo.

2.1. Codificación

Cómo se ha dicho anteriormente, se codificarán en arrays bidimensionales, cuyas filas y columnas mantendrán el significado del lenguaje natural. Como lenguaje de programación se ha escogido *Python* y para las operaciones con matrices, la librería *numpy*.

Cada individuo estará formado por una posible solución, sin necesidad de ser la óptima, y el contenido genético de este serán todos y cada uno de los valores que tiene en sus 81 celdas, que será el que se utilizará para analizar el nivel de adaptación y para el cruce entre individuos.

2.2. Población inicial

La población inicial se obtiene generando un número de individuos dado por el tamaño de población que se escoja. Cada individuo se genera añadiendo números aleatorios a las casillas vacías del sudoku inicial. Para reducir el coste computacional y el tiempo de ejecución, se rellena cada bloque cumpliendo la restricción de bloque del sudoku. Es decir, cada bloque se rellena con una permutación de números del 1 al 9, asegurando así que los bloques incluyen todos los números posibles, comenzando todos los individuos con un *fitness* mayor que si no se tuviera en cuenta dicha restricción. Además, esto nos permite que ara el cálculo del *fitness* sólo sea necesario evaluar las columnas y las filas.

2.3. Fitness y funcion fitness

El *fitness* es un atributo derivado de cada sudoku o individuo, que constituye la parte más importante del algoritmo. Esto es así debido a que la selección de soluciones adecuadas y la separación entre individuos más y menos aptos la determina dicho atributo. La elección de la función de *fitness* es un paso trascendental. Una mala función no permitirá llegar a la solución óptima.

Para determinar dicha función, hay que tener en cuenta como se crea un

individuo. De esta manera, solo se evaluará la cantidad de números sin repetir en columnas y filas, puesto que se cuidará que no haya repetidos en los bloques.

La función *fitness*, por tanto, vendrá dada por:

$$\begin{array}{c} \sum_{i=1}^{Nfil} \text{Cantidad números no repetidos en bloque } i \\ + \sum_{i=1}^{Ncol} \text{Cantidad números no repetidos en columna } i \end{array}$$

Siendo $Nfil, Ncol$, el número de bloques y de columnas del sudoku, respectivamente, que serán ambos de valor 9 para un sudoku clásico.

El máximo valor de la función se alcanza cuando la función vale 162, ya que si no hay números repetidos en filas, cada fila tendrá 9 números distintos, y lo mismo ocurrirá con las columnas. Cuando se alcance este valor de función *fitness*, se habrá obtenido la solución óptima y el algoritmo se detendrá.

Para que la función de adaptación siga funcionando a lo largo de las generaciones, hay que mantener un cuidado sobre cómo se generan los descendientes y las mutaciones, tratando de no cambiar un número en alguna permutación y provocar repetidos.

2.4. Selección

La selección lleva a cabo el concepto de *selección natural*, facilitando la supervivencia y por tanto la reproducción de los individuos con mayor adaptación, que será el valor de la función *fitness*.

En este algoritmo la selección se lleva a cabo mediante un ranking lineal, ordenando a los individuos según su nivel de adaptación y asignando a cada uno un valor descendiente lineal de probabilidad de ser seleccionado para reproducirse.

2.5. Recombinación

La recombinación es una parte destacada en el desarrollo del algoritmo para la búsqueda de la solución objetivo. Permite evolucionar recombinando los genes o los valores de las casillas de un individuo. En la solución que se presenta, la recombinación se lleva a cabo en dos pasos:

1. Seleccionando a dos progenitores. Según la probabilidad de ser seleccionados, que ha sido enunciada con anterioridad.

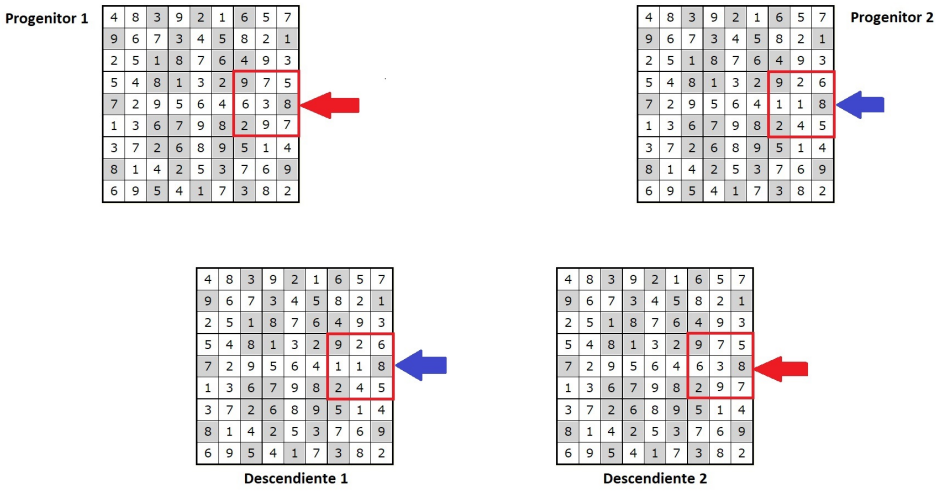

Figura 1: Ilustración de un cruce en el que se intercambia un bloque señalado en rojo.

2. Intercambio de información. El intercambio de información genética se lleva cabo mediante intercambio de bloques entre dos individuos progenitores. Esto garantiza que los bloques siempre sean una permutación de los 9 números posibles. De intercambiarse filas o columnas, podría provocarse algún número repetido. Además, se eligen los bloques que han de intercambiarse aleatoriamente, pudiendo resultar intercambiados hasta 4 bloques (intercambiar 5 es equivalente a intercambiar 4 al ser un total de 9 bloques).

Los cruces ocurren con una probabilidad determinada. En caso de no ocurrir, los progenitores seleccionados seguirán en la población en lugar de ser sustituidos por sus descendientes.

2.6. Mutación

El proceso de mutación provoca cambios en la información genética de un individuo. Sucede con una probabilidad especificada, normalmente pequeña. Para este caso, la función de mutación conlleva algunos aspectos importantes.

En primer lugar, no se pueden mutar los valores proporcionados en el sudoku objetivo, pues de ser así se complicaría el alcance de una solución óptima.

Por otro lado, para mantener los bloques sin repeticiones, la mutación no se lleva a cabo cambiando un valor de una celda por un número aleatorio, sino que se realiza con un intercambio por otro número del mismo bloque.

. Sólo ocurrirá un intercambio cada vez que ocurra la mutación.

2.7. Perturbación

Cuando una población pase varias generaciones sin evolucionar, es decir, con el mismo individuo siendo el de mayor adaptación, se entenderá que se ha alcanzado un máximo local y se procederá a realizar de nuevo una permutación de los nueve posibles dígitos para todos los individuos de la población. Así se ayudará a salir de ese máximo local, realizando la permutación de la misma manera que al crear un individuo, con la salvedad de que solo se realizará en un bloque para cada sudoku.

2.8. Desarrollo

El desarrollo del algoritmo es un proceso iterativo que repite las funciones anteriormente explicadas. El principal problema de este tipo de algoritmos es que pueden quedarse atrapados en un máximo local si los individuos han convergido a un mismo valor, generalmente cercano pero menor que la solución óptima. Para tratar con este problema, se ha añadido al desarrollo un control sobre cuántas generaciones lleva sin cambiar el individuo de mayor *fitness*. Si este numero de generaciones alcanza una antigüedad determinada con anterioridad, se perturbarán los individuos de la población y se incrementará la probabilidad de mutación.

2.8.1. Pseudocódigo

```
Generar población inicial aleatoriamente
repeat
  Ordenación de la población
  Añadir a nueva población los sudokus mas aptos
  repeat
    if Poblacion envejecida then
      Generar individuos aleatoriamente y añadir a nueva población
    else
      Selección de dos progenitores
      Intercambio de información entre ellos
      Mutación de descendientes
```

Añadir a la nueva población los descendientes
end if
until Se obtenga una población completa
Sustituir población antigua por nueva
until Se haya alcanzado la solución o un número determinado de generaciones

2.8.2. Costes

El análisis del coste de este tipo de algoritmos suele ser complicado ya que no son lineales, no se sabe con exactitud si llegarán a la solución ni cuánto tardarán en llegar a ella. Esta evolución esta fuertemente ligada a la multitud de parámetros que requiere el algoritmo, que se han ido enunciando a lo largo del informe y que son los siguientes:

Tamaño del sudoku. El tamaño del sudoku, determinado por el número de filas, columnas y bloques, afecta al rendimiento del algoritmo y se denotará con n. No obstante, al tratarse de un N pequeño, hay que considerar que costes de funciones de n no serán elevados en la práctica.

Tamaño de las poblaciones. La cantidad de individuos que tiene una población, denominado con P.

Número de élites. Cantidad de individuos de una población que serán elegidos como superiores en base a su adaptación.

Probabilidad de recombinación. Probabilidad de que dos progenitores produzcan dos descendientes. De no cumplirse, los progenitores continuarán en la población.

Antigüedad. Número de generaciones a partir de la cual se regenerarán los individuos peor adaptados.

Probabilidad de mutación. Probabilidad de que un individuo mute.

La elección del valor de dichos parámetros es experimental. Mas adelante se mostrará una serie de pruebas para diferentes valores de los parámetros.

Sin embargo, sí se puede hacer una análisis del coste de las funciones de dicho algoritmo:

Crear Individuo

Require: $Objetivo = matriz\ nxn$
Require: Celda con valor cero en $Objetivo$ indica celda vacía
  for $i = 0$ hasta n incremento 3 **do**
    for $j = 0$ hasta n incremento 3 **do**
      $posibles = [1, 2, 3, 4, 5, 6, 7, 8, 9]$
      if $objetivo[i, j] == 0$ **then**
        $numero = random(posibles)$
        while numero $not\ Valido$ **do**
          $numero = random(posibles)$
        end while
        Eliminar $numero$ de $posibles$
        $individuo[i, j] = numero$
      end if
      ...recorrer todas las celdas...
      if $objetivo[i + 2, j + 2] == 0$ **then**
        $numero = random(1, 9)$
        while $numero\ not\ Valido$ **do**
          $numero = random(1, 9)$
        end while
        Eliminar $numero$ de $posibles$
        $individuo[i + 2, j + 2] = numero$
      end if
    end for
  end for

Validez

  for $i = 0$ hasta n **do**
    if $individuo[i, j] == numero$ **then**
      return FALSE
    end if
  end for
  return TRUE

El coste de la función de creación en el peor caso será $O(n^5)$. Esto se debe a que en el peor caso, realizar la operación de eliminar un número de *posibles* requiere recorrer posibles con un coste de $O(n)$. Esto se realiza n veces, una para cada bloque, lo que provoca coste $O(n^2)$. Lo mismo ocurre con la llamada a la función $Validez$, que tiene coste de orden n, de donde se obtiene $O(n^3)$. Todas estas operaciones ocurrirán en dos bucles for desde

0 hasta $n/3$, luego dos bucles de coste $O(n/3)$, lo que lleva a un total de $O(n^5)$.

Fitness

for $i = 0$ hasta n **do**
  Eliminar repetidos fila i
  Eliminar repetidos columna i
  $Fitness$ = Suma longitud de fila y columna i
end for

Se trata de una función que realiza dos eliminaciones de repetidos, lo que tiene un coste de la longitud de la fila (n) en el peor caso, además de una suma en un bucle desde 0 hasta n. Su coste es por tanto $O(n^2)$.

Cruce

Require: Ocurre con probabilidad dada
  Copia progenitores
  $NumeroIntercambios = random(0, 4)$
  for $i = 0$ hasta $NumeroIntercambios$ **do**
    $Intercambio$ entre copia progenitores
  end for
  return Copias

En esta función, es necesario tener una copia de los progenitores para que sean ellos los descendientes en caso de no producirse el cruce,lo cual provoca un coste de orden n^2. El intercambio se realiza seleccionando un bloque aleatorio e intercambiándolo entre los progenitores. Son operaciones elementales, por lo que el coste global en el peor caso es el máximo número de intercambios, $O(4)$. Resulta un coste total de $O(n^2)$.

Mutacion

Require: Ocurre con probabilidad dada
  $NumeroMutaciones = random(0, n)$
  for $i = 0$ hasta $NumeroMutaciones$ **do**
    $Mutacion$
  end for

La mutación se lleva a cabo con una serie de operaciones elementales como elección aleatoria del bloque e intercambio de dos de sus casillas. Su orden es por tanto en el peor caso $O(n)$.

Perturbar

```
for i = 0 hasta P do
  posibles = [1, 2, 3, 4, 5, 6, 7, 8, 9]
  if objetivo[i, j] == 0 then
    numero = random(posibles)
    while numero not Valido do
      numero = random(posibles)
    end while
    Eliminar numero de posibles
    individuo[i, j] = numero
  end if
  ...recorrer todas las celdas...
  if objetivo[i + 2, j + 2] == 0 then
    numero = random(1, 9)
    while numero not Valido do
      numero = random(1, 9)
    end while
    Eliminar numero de posibles
    individuo[i + 2, j + 2] = numero
  end if
end for
```

En la función que perturba todos los individuos de la población se realizan los mismos pasos que en la creación de un individuo, con la diferencia de que sólo se opera sobre un bloque y no sobre todos. Esto da un orden de $O(n^3)$ al eliminar los dos bucles for de incremento 3. Además, hay que realizar esto para toda la población, de tamaño P. Se tiene por esto un coste $O(n^3 * P)$.

Conocido el coste de cada función, se puede realizar un análisis del coste del desarrollo de todo el algoritmo:

Resolución

```
Require: Tamaño poblacion P conocido
Require: Numero generaciones G conocido
Require: Antigüedad máxima permitida A conocida
Require: Cantidad de elites NE conocida
  for i = 0 hasta P do
    Crear Individuo
    Añadir Individuo a Poblacion
  end for
  for i = 0 hasta G do
```

```
    for j = 0 hasta P do
      if Fitness == 162 then
        return SOLUCION
      end if
    end for
    Ordenar Poblacion
    Mejor Adaptacion = Primer Individuo
    if Antigüedad = A then
      NuevaPoblacion = Perturbar Poblacion
    else
      for k = 0 hasta NE do
        Añadir individuo a NuevaPoblacion
      end for
      for l = NE hasta P incremento 2 do
        Seleccion de dos progenitores
        Cruce de los progenitores
        Mutacion de descendientes
        Añadir descendientes NuevaPoblacion
      end for
    end if
    Ordenar NuevaPoblacion
    if Mejor Adaptacion=Primer Individuo then
      Antigüedad++
    else
      Antigüedad= 0
    end if
    Poblacion = NuevaPoblacion
  end for
```

Costes:

1. En primer lugar se genera una población inicial. Se crea un individuo P veces, y se añade a la población (coste 1). Por lo tanto, el primer bucle *for* provoca un coste de orden $P * n^5$.

   A partir de aquí comienza un bucle hasta G en el que se realizan varios pasos:

2. Bucle *for* en busca de un individuo óptimo. El coste de la función Fitness P veces da un coste $O(P * n^2)$.

3. Ordenar población (con coste de $O(n * log(n))$), en función del fitness.

Por tanto el coste total es $O(n^3 * log(n))$.

4. Si la condición del if es cierta, el coste será el de la función $Perturbar$, $O(n^3 * P)$. De no darse la condición, el primer for tiene un coste de orden NE, que en el peor caso NE sera igual a P, luego $O(P)$. En el segundo, la selección es inmediata, el cruce tiene coste de orden $O(n^2)$ y la mutación tiene coste $O(n)$. Todo ello repetido $(P - NE)/2$ veces, provoca un coste de $O(n^2 * (P - NE))$. Como en este ultimo, el peor caso se da cuando NE es igual a cero, estos dos bucles suman un coste $O(n^2 * (P))$.

   El coste de la sentencia if total será de orden $O(n^3 * P)$.

5. Ordenar población de nuevo, $O(n^3 * log(n))$.

6. Asignación a $Poblacion$ de $NuevaPoblacion$, coste de orden $O(P * n^2)$, por estar formada por matrices nxn.

Para realizar el análisis del coste, se ha considerado el tamaño del sudoku debido a la posibilidad de tratar de resolver sudokus que no sean matrices $9x9$ (nxn). Para el caso que se trata, n siempre tendrá valor 9, por lo que todos los costes en los que aparece implicado n se reducirán a cambiar n por una constante. De esta manera, se puede resumir el coste global del algoritmo en términos asintóticos a (para los pasos anteriores):

1. $O(P * cte)$
2. $O(P * cte)$
3. $O(cte)$
4. $O(P * cte)$
5. $O(cte)$
6. $O(P * cte)$

Cómo del paso 2 al 6 se repite G veces, se simplifica a un coste asintótico de $O(G * P)$.

2.8.3. Pruebas

En la práctica el algoritmo no resultó eficiente. En general se comportó mejor para tamaños de población no extremadamente grandes y valor de antigüedad pequeño, ya que esto provocaba muchas permutaciones en las

generaciones. Los valores que mejor parecieron funcionar fueron seleccionar como élites al 5 % de la población, y probabilidad de recombinación de 0,99.

Para estos valores, se comentan algunos de los resultados:

- Para tamaño de población 100, fueron necesarias 530 generaciones para obtener la solución. El tiempo total fue de 36,71 segundos.

- Para tamaño de población 3000, se llegó a un resultado cercano a la solución inicial en la generación 57, el cual resultó máximo local y el proceso llegó al peor caso alcanzando el número de generaciones máximo. Puede destacarse como el tiempo de pasar de una generación a otra resulta mucho mayor, pues se consiguieron 57 generaciones en aproximadamente 62,7 segundos.
  Para los mismos valores, en otra ocasión, se alcanzó la solución optima en la generación 349, tras 108 segundos.

- En otra ocasión, reduciendo el tamaño de población a 1000, se alcanzó un valor muy cercano a la solución en la generación 99, el cual no cambió hasta la 210 cuándo llegó a la solución en un tiempo de 85,234 segundos.

- Para un mismo tamaño de población, no aumentaba el tiempo de cambio de generación al variar el resto de parámetros. En otra prueba con los mismos valores no se obtuvo la solución óptima.

En resumen, el aumento de tamaño de la población provocaba, como era de esperar, un aumento notable de tiempo de cambio entre generaciones, y por tanto un aumento de tiempo para llegar a la solución óptima. No obstante, este aumento no era lineal debido a que un número de individuos mayor permite una mayor combinatoria entre sus genes, y aumenta la probabilidad de llegar a mejores soluciones. Sin embargo, en muchas ocasiones la espera era demasiado larga sin mejorar desde máximos locales.

3. Conclusiones

Los algoritmos genéticos resultan útiles para problemas con espacios de búsqueda complejos, que tengan muchas posibles combinaciones el cual es el caso de los sudokus. Un algoritmo genético no realiza una búsqueda clásica, sino que realiza una búsqueda "guiando"la aleatoriedad, tratando de reducir el espacio de búsqueda. Teóricamente podría resultar interesante, por tanto, aplicar este método de resolución a los sudokus. Sin embargo, en la práctica, se atascaba en máximos locales frecuentemente y no era posible resolver el sudoku en un tiempo asequible. Esto, en conocimiento de otros algoritmos de resolución de sudokus que son capaces de resolverlos en unos pocos segundos, plantea una clara desventaja para los genéticos. Sin embargo, se ha probado que este algoritmo alcanza una solución muy cercana a la óptima en muchas ocasiones con una rapidez considerable, lo cual puede resultar interesante en otro tipo de problemas que no requieran una solución tan específica como la requerida por un sudoku.

El desempeño del algoritmo esta muy fuertemente ligado a los parámetros de este y encontrar una combinación óptima para todos los parámetros puede resultar complicado.

Con todo, se concluye que la utilización de algoritmos genéticos en la resolución de sudokus no es la decisión mas adecuada si se trata de reducir el tiempo de ejecución. Si por excepción se tratase de encontrar una solución cercana a la correcta, entonces el algoritmo puede ser adecuado por la rapidez de mejora en las poblaciones iniciales.

Referencias

[1] Sudoku, Wikipedia: $https://es.wikipedia.org/wiki/Sudoku$

[2] Complejidad temporal de funciones de Python: $https://wiki.python.org/moin/TimeComplexity$

[3] Genetic Algorithms and Sudoku, Dr. John M. Weiss : $http://micsymposium.org/mics_2009_proceedings/mics2009_submission_66.pdf$

[4] Algoritmos Genéticos: $http://eddyalfaro.galeon.com/geneticos.html$