

Algoritmos Genéticos para la Resolución de Sudokus

Luciano Bello

111863-8

luciano (at) linux.org.ar

Inteligencia Artificial

Profesora: María Florencia Pollo

Índice

1. Enunciado	2
2. Introducción, Definiciones y Convenciones	2
2.1. El cromosoma	3
2.2. El flujo del algoritmo	3
2.2.1. Generación de la población inicial	3
2.2.2. Selección - Función de aptitud	4
2.2.3. Cruzamiento	4
2.2.4. Mutación	5
2.2.5. Condición de finalización	5
3. Pruebas y Análisis Paramétrico	5
3.1. Análisis 1: Tamaño de la población y cantidad de padres	6
3.2. Análisis 2: Elección del algoritmo de cruzamiento	7
3.3. Análisis 3: Cantidad de incógnitas del enunciado	9
4. Conclusión Final	10

1. Enunciado

Se solicita definir un problema, que sea apropiado para ser resuelto con Algoritmos Genéticos, para luego implementarlo y ejecutarlo para encontrar la combinación de parámetros que haga el mejor resultado en la menor cantidad de generaciones.

2. Introducción, Definiciones y Convenciones

El problema que se propone abordar en este trabajo práctico es la resolución automatizada del juego Sudoku¹. Sudoku es un pasatiempo que se popularizó en Japón en 1986, aunque es originario de Estados Unidos, y se dio a conocer en el ámbito internacional en el 2005[3]. El objetivo es rellenar una cuadrícula de 9x9 celdas (81 casillas en total) subdividida en sectores de 3x3 (también llamadas *cajas* o *regiones*) con cifras de 1 a 9 partiendo de algunos números ya dispuestos en algunas de las celdas. No se requiere operar con los números y podrían ser reemplazados por colores, letras, figuras o cualquier conjunto de nueve elementos diferenciados. El motivo de usar números es que se memorizan mejor. No se debe repetir ninguna cifra en una misma fila, columna o región. Un sudoku está bien planteado si la solución es única[1].

El programa que aborda el problema acompaña a este documento y se llama `miSudoku.py`. Está desarrollado en Python y en el archivo `README.txt` se encuentran las instrucciones para correrlo en diferentes sistemas operativos. Además, se adjunta información adicional como ser los gráficos de todos los análisis y casos de pruebas.

¹del japonés *número único* (su=número, duko=único)

2.1. El cromosoma

Las soluciones candidatas (población) es una cadena de valores de largo n , donde n es la cantidad de celdas incógnita. Cada posición de la cadena es un valor entre 1 y 9, de manera que las soluciones (individuo) coinciden con la siguiente expresión regular:

$$[1-9]^n.$$

Existe una ineficiencia intrínseca en manejar los cromosomas a nivel bit. Como se necesita representar 9 valores posibles en cada celda, la cantidad mínima de bits necesario es de 4. Esto genera 16 (2^4) valores posibles, de los cuales solo se utilizaría el 56%, teniendo que descartar el resto. Esta es la razón por la que se considera atómico al valor de la celda (gen).

Dado que las soluciones se representan como una larga cadena, se considera genotipo a las diferentes porciones de ésta que completan una fila del sudoku dado. Nótese que un enunciado distinto impacta en el tamaño de cada genotipo. Por ejemplo, dado el sudoku de la figura 1, el cromosoma esta compuesto por 53 genes que completan las 53 incógnitas. El primer genotipo está formado por los primeros 4 genes del cromosoma porque la primer fila tiene 4 celdas vacías. Análogamente, el segundo genotipo está compuesto por los siguiente 7 genes y así siguiendo.

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

Figura 1: Ejemplo de sudoku con 53 celdas incógnitas

2.2. El flujo del algoritmo

En la figura 2 se representa el flujo del algoritmo a utilizar. A continuación se detalla cada una de las funciones.

2.2.1. Generación de la población inicial

El parámetro puede ser modificado en la variable `population` del archivo de configuración `sudoku.conf`. La población inicial se genera mediante la función `genPopulation(n)`, donde n es la cantidad de individuos a generar. Basándose en la cantidad de celdas incógnita como largo, crea cromosomas con valores al azar y entre 1 y 9 para cada gen. La cantidad de individuos se mantiene constante durante toda la ejecución.

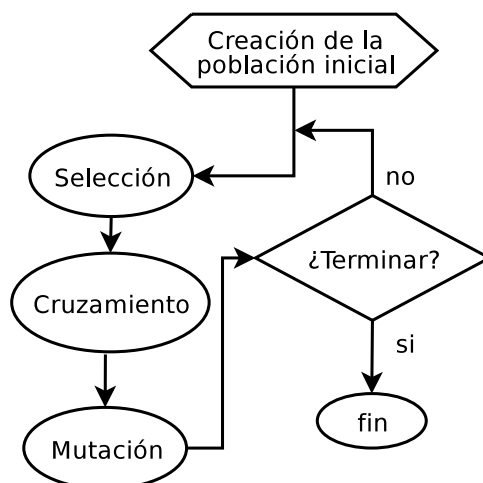


Figura 2: Flujo del algoritmo genético

2.2.2. Selección - Función de aptitud

La selección se realiza en base a la valorización de la solución según la función de aptitud. La misma está definida en el código como `funcion_de_aptitud(m)`, donde `m` es el individuo a evaluar. Para obtener este valor se suman la cantidad de repeticiones en todas las filas, columnas y regiones. **A menor valoración, más apto el individuo.**

La población es ordenada luego del más apto al menos. En caso de empate, se reorganizan al azar. De la cima de esta lista ordenada se toma la población elegida para el cruzamiento según la variable `selected_patterns` del archivo `sudoku.conf`.

2.2.3. Cruzamiento

Con los individuos seleccionados de la parte superior de la población ordenada se procederá al cruzamiento. Tomados de a pares, cada pareja generará dos nuevos individuos que reemplazarán a los menos aptos de la población. Así la máxima cantidad de cromosomas elegidos como padres es la mitad de la población total. Los métodos de cruzamiento que pueden definirse son:

1. cruzamiento simple: Una parte de la madre, otra del padre. Corta el cromosoma en dos al azar.
2. binomial puro: No tiene en cuenta los genotipos. Usa genes del padre y de la madre al azar y en orden.
3. binomial impuro: Corta teniendo en cuenta los genotipos. Toma genotipos de la madre o del padre al azar y en orden.
4. multipunto: Corta teniendo en cuenta los genotipos. Toma genotipos de la madre o del padre de forma alternada.
5. operar: Para producir más variedad se opera con los genes sumándolos, restándolos o multiplicándolos.

Los algoritmos a usar son seleccionados por medio de la variable `crossover_algorithms` del archivo de configuración y van entre corchetes, referenciados por su número o nombre y separados por comas. Si se requiere utilizar más un algoritmo que otro, se lo puede repetir. Se selecciona al azar con distribución uniforme. Por ejemplo, si `crossover_algorithms=[1,1,5]` el algoritmo de cruzamiento *simple* se usará en 2 de cada 3 casos y el algoritmo *operar* las veces restantes.

2.2.4. Mutación

La función `mutarIndividuo(x)`, donde `x` es el individuo a mutar, se corre al azar, con una distribución uniforme dada por la variable `mutation`. Los individuos que pueden mutar son aquellos no seleccionados para padres y no hijos. Se toman al azar y se modifica alguno de sus genes de forma aleatoria.

2.2.5. Condición de finalización

Dado que el objetivo, más allá de solucionar el sudoku, es estudiar el comportamiento y evolución de la población, el programa se detendrá cuando se hayan hecho 300 iteraciones. Luego se imprimirá la solución más apta. **Si su valorización es 0, entonces es la respuesta al sudoku.**

3. Pruebas y Análisis Paramétrico

Se realizaron 3 tipos de pruebas, cada una enfocada a un aspecto distinto del algoritmo. Las pruebas se ejecutaron 3 veces independientes en diferentes máquinas. Los resultados de estas pruebas están almacenados en el directorio `pruebas/corridas` y puede ser consultados para más información. Los gráficos generados por las mismas se encuentran almacenados en `pruebas/graficos`. Estos gráficos se expresan en función de la cantidad de generaciones. Como las evoluciones tienden a estabilizarse (los gráficos que incluyen las 300 vueltas se encuentran en `pruebas/graficos/completos`) solo se reproducen las primeras iteraciones. En cada caso se representa en forma de barra los valores más bajos y más altos obtenidos para esa muestra. La línea que cruza el bloque lo hace por el promedio de los datos obtenidos en las diferentes corridas.

En todos los casos se mutó una de cada 10 veces.

De cada ejecución se obtuvieron como resultado las siguientes caracterizaciones:

- Población única: La cantidad de individuos de la población sin contar los individuos iguales.
- Mejores de la media: Dado la valorización media, que cantidad de individuos está por encima de ésta.
- Aptitud media: La valorización promedio de toda la población.
- El mejor: La valorización del individuo más apto.
- El peor: La valorización del individuo menos apto.
- El 20 % mejor: La valorización promedio del 20 % de la población más apta.

3.1. Análisis 1: Tamaño de la población y cantidad de padres

Este análisis busca estudiar el efecto del tamaño de la población y de la cantidad de individuos elegidos para padres de la siguiente generación. Para esto, se intentó resolver el sudoku de la figura 3 y utilizando el método de cruzamiento *simple* el 67% de las veces y *operar* el 33% restante.

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

Figura 3: Enunciado propuesto para el análisis paramétrico 1

Con respecto a los parámetros sobre la población se hicieron 4 variaciones:

1. Población de 500 individuos y seleccionando los mejores 50 (`population=500, selected_patern`
2. Población de 500 individuos y seleccionando la mitad más apta (`population=500, selected_patterns=population/2`)
3. Población de 200 individuos y seleccionando los mejores 50 (`population=200, selected_patern`
4. Población de 200 individuos y seleccionando la mitad más apta (`population=200, selected_patterns=population/2`)

Los resultados finales, después de 300 iteraciones, son:

Caso	población única	mayores a la media	a aptitud me- dia	el mejor	el peor	el 20% me- jor
1.1	15 20 16	14 18 14	42 35 34	41 33 33	104 98 98	41 33 33
1.2	89 83 85	96 96 88	36 26 40	25 12 30	99 108 101	25 12 30
1.3	16 16 11	14 14 10	43 42 38	40 39 36	99 101 102	40 39 36
1.4	35 34 45	34 34 36	35 42 37	24 34 27	104 108 102	24 34 27

Cada uno de los 3 valores corresponde a cada una de las corridas. En las figuras 4 puede verse el promedio de cada caso (a excepción de *el peor*). Nótese que los mejores resultados se consiguieron en los casos 1.2 y 1.4, donde la cantidad de individuos seleccionados para la siguiente generación fue la mitad de la población total. También puede observarse una leve mejoría cuando la población es mayor, como en los casos 1.1 y 1.2. Puede deducirse que una mayor variedad de individuos únicos (en rojo de la figura

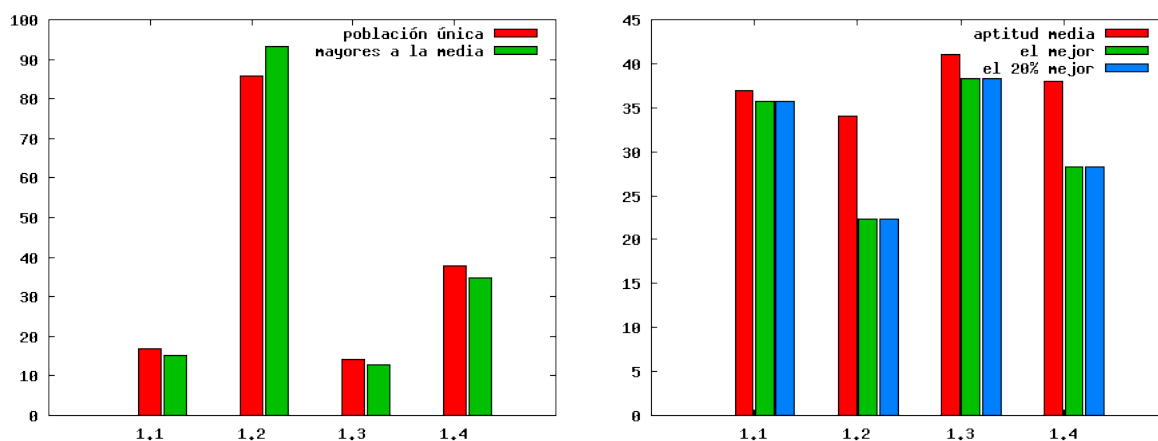


Figura 4: Análisis 1 de la población y su aptitud en el estado final (promedio de las corridas)

4 de la izquierda) permite alcanzar mejores resultados. La figura 5 ilustra la evolución durante las primeras 100 iteraciones. Obsérvese cómo el caso 1.2 mejora la aptitud (es decir, se hace más baja) cuando logra estabilizar la variedad de individuos, entre la vuelta 30 y 60.

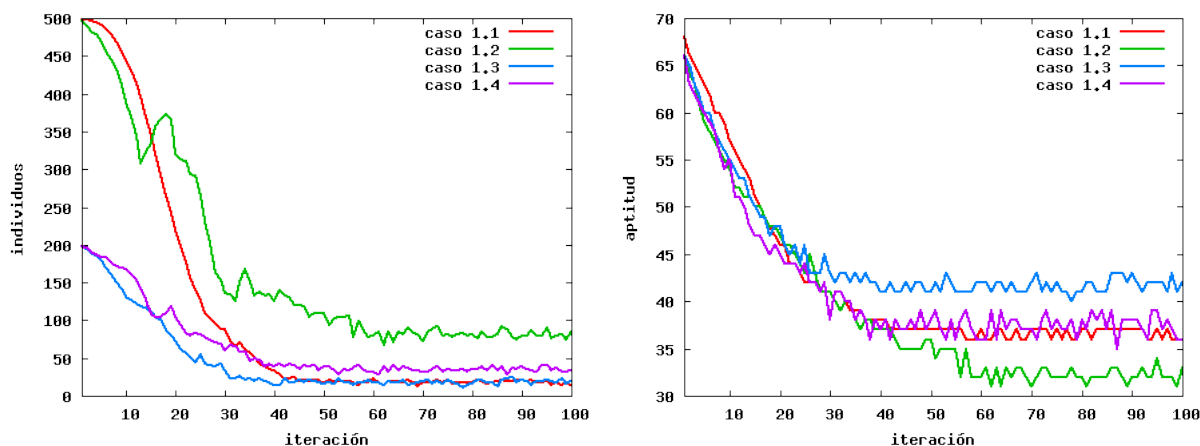


Figura 5: Evolución de la población única y de la aptitud media en el análisis 1

3.2. Análisis 2: Elección del algoritmo de cruzamiento

Para este análisis se utilizará el mismo enunciado de la figura 3 con una población de 500 individuos y seleccionando la mitad más apta (`population = 500`, `selected_patterns = population/2`). Se estudiará el efecto que produce el algoritmo de cruzamiento en el resultado y evolución de la población.

Para ello se utilizarán las siguiente configuraciones:

1. Utilizando únicamente métodos de transposición que ignoran los genotipos: *cruzamiento simple* y *cruzamiento binomial puro* (`crossover_algorithms=[1,2]`)

2. Utilizando únicamente métodos de transposición que tengan en cuenta los genotipos: *cruzamiento binomial impuro* y *cruzamiento multipunto* (`crossover_algorithms=[3,4]`)
3. Utilizando únicamente el método *operar* que realiza operaciones matemáticas sobre los padres, haciendo que los hijos no se les parezcan (`crossover_algorithms=[5]`)
4. Utilizando una combinación entre transposición y operados matemáticamente en proporciones desiguales: 67% de las veces *cruzamiento simple* y 33% *operar* (`crossover_algorithms=[3,4,5]`)

Los resultados finales, después de 300 iteraciones, son:

Caso	población única	mayores a la media	aptitud media	el mejor	el peor	el 20% mejor
2.1	1 29 4	0 20 0	6 16 11	6 16 11	6 20 11	6 16 11
2.2	1 1 1	0 0 0	30 30 24	30 30 24	30 30 24	30 30 24
2.3	481 485 477	233 231 234	63 63 63	48 49 49	89 94 92	53 53 54
2.4	110 109 86	104 88 86	34 30 26	22 18 14	104 104 106	22 18 14

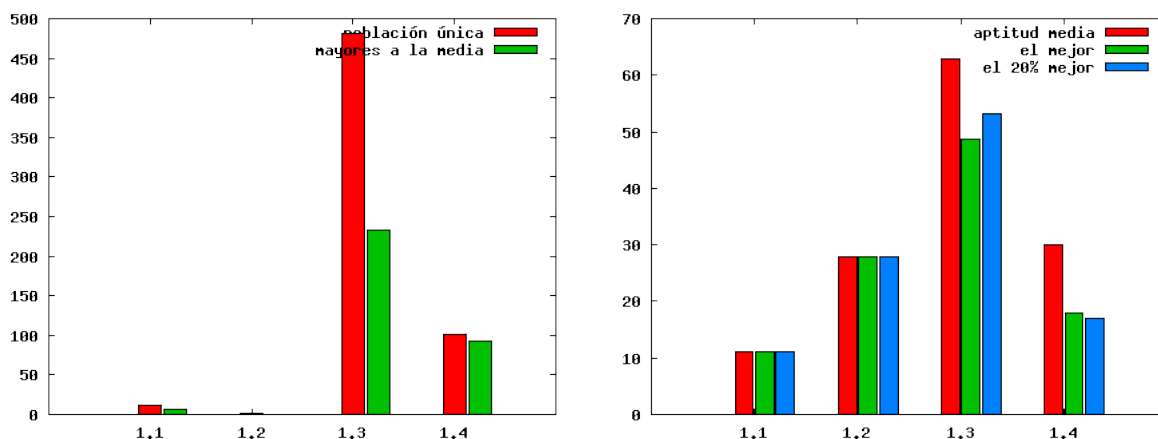


Figura 6: Análisis 2 de la población y su aptitud en el estado final (promedio de las corridas)

En la figura 6 de la izquierda puede observarse como los algoritmos de cruzamiento por transposición (los casos 2.1 y 2.2) terminaron con poblaciones extremadamente homogéneas, condición que se consolidó totalmente en la iteración 50 (ver figura 7). El cruzamiento *operar* generó poblaciones con altos índices de diferenciación, pero sus resultados en términos de adaptabilidad no fueron buenos. El mejor de los individuos mejoró de forma depreciable durante las sucesivas iteraciones.

A su vez, los algoritmos que tuvieron en cuenta los genotipos tampoco resultaron particularmente eficientes y el caso de prueba 1.2 agotó toda posibilidad al converger la totalidad de la población a una única solución, hacia la iteración 35. El hecho de transponer bloques más grandes solo provocó el adelantamiento de la convergencia en casi 20 vueltas, como puede observarse en la figura 7 de la izquierda de la comparación de los casos 2.1 y 2.2.

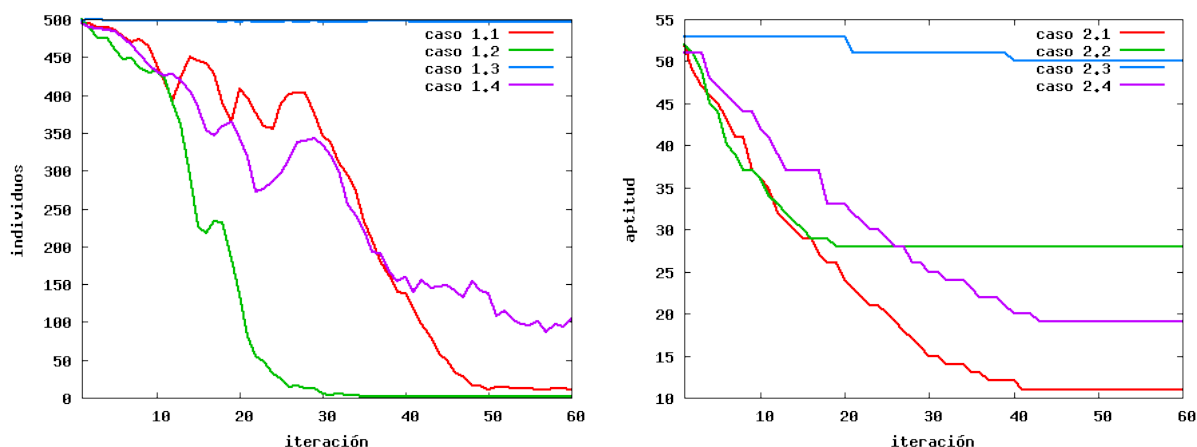


Figura 7: Evolución de la población única y del individuo más apto en el análisis 2

De manera totalmente opuesta al análisis 1 (ver sección 3.1), mayor heterogeneidad en la población no solo no aseguró mejores resultados, sino que la adaptabilidad resultó ser inversamente proporcional a la variedad de individuos. Así mismo, el caso 2.1 elaboró buenos resultados mientras la población se mantuvo diferenciable pero se asentó cuando la tasa de respuestas gemela alcanzó el 70% en la vuelta 40.

Por otra parte, la opción 2.3 tuvo un comportamiento diametralmente distinto, generando desigualdad entre los individuos, pero no mejorando en absoluto su calidad. Un mejor balance puede observarse en 2.4, cuando se utilizó una combinación de los algoritmos de transposición con el de operación matemática, el cual mantuvo una diversidad cercana al 20% de la población y, aunque su mejora en la calidad se vio estancada, fue el último en detener su progreso.

3.3. Análisis 3: Cantidad de incógnitas del enunciado

En 1986, Nikoli² introdujo como consigna en el armado de sudokus que el número de cifras que dadas esté restringido a un máximo de 30[2]. Esto significa que las incógnitas sean, como mínimo, 51. En el próximo y último análisis se flexibilizará dicha restricción, con el objetivo de facilitarle el trabajo al algoritmo. Para éste análisis se utilizarán a una población inicial de 500 individuos y seleccionando la mitad más apta (`population = 500`, `selected_patterns = population/2`). El algoritmo de cruzamiento será la combinación estudiada en el caso 2.4 del análisis anterior, compuesto por 66% de las veces de *cruzamiento simple* y el 33% del *cruzamiento operar* (`crossover_algorithms=[1,1,5]`). El enunciado es el mismo sudoku, aunque con diferentes niveles de completitud, a saber:

1. con 45 incógnitas (representado en la figura 8)
2. con 40 incógnitas (representado en la figura 9)
3. con 35 incógnitas (representado en la figura 10)

²editor japonés que se especializa en juegos, y especialmente, en enigmas de lógica, dueño de una editorial homónima que importó el Sudoku a Japón en 1984

4. con 30 incógnitas (representado en la figura 11)

5. con 25 incógnitas (representado en la figura 12)

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

Figura 8: caso 3.1

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

Figura 9: caso 3.2

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

Figura 10: caso 3.3

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

Figura 11: caso 3.4

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

Figura 12: caso 3.5

Los resultados finales, después de 300 iteraciones, son:

Caso	población única	mayores a la media	a	aptitud me- dia	el mejor	el peor	el 20% me- jor
3.1	106 79 74	104 80 74		42 35 28	30 24 17	110 109 112	30 24 17
3.2	67 102 99	66 90 88		31 30 35	24 19 26	98 96 98	24 19 26
3.3	58 80 85	64 80 88		18 27 22	11 19 11	79 83 83	11 19 11
3.4	85 87 84	86 88 84		18 19 17	9 10 8	73 76 72	9 10 8
3.5	83 83 85	84 86 88		12 10 13	3 0 4	68 66 66	3 0 4

Tal como podía preverse, a menor cantidad de incógnitas, mejores soluciones. Los resultados y progresos referidos a la población se encuentran parejos en todos los casos. La evolución de la población se mantuvo constante y su mejora se realizó con la misma intensidad de forma pareja. Por otra parte, las pruebas 3.1 y 3.2 finalizan muy superpuestas (ver figura 14 a la derecha), lo que puede significar que no hay una real diferencia entre 40 y 45 incógnitas o que el azar así lo dispuso.

Por primera vez, una de las corridas logró alcanzar una solución, durante el caso 3.5. Se trata de pruebas/corridas/mariano_3_5.txt, y lo hizo de forma relativamente temprana, durante la iteración 49 (ver figura 15)

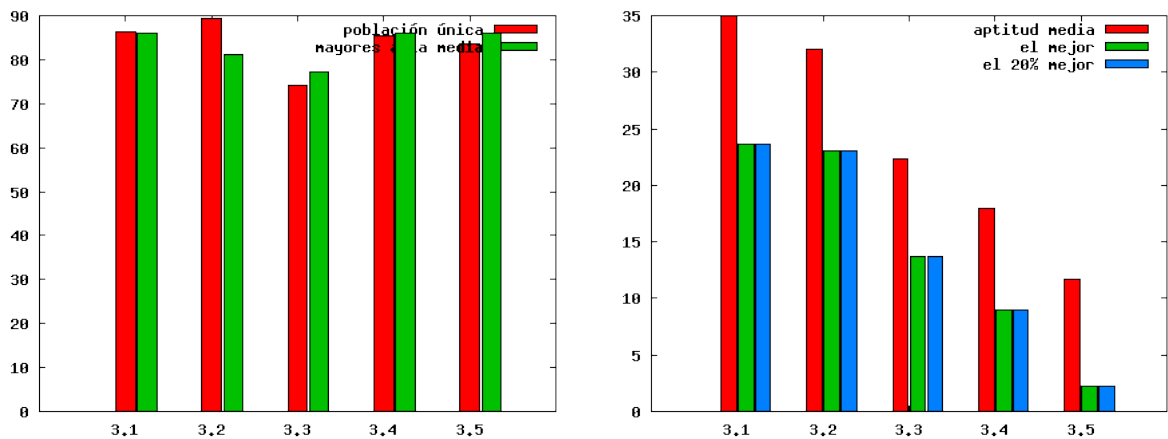


Figura 13: Análisis 3 de la población y su aptitud en el estado final (promedio de las corridas)

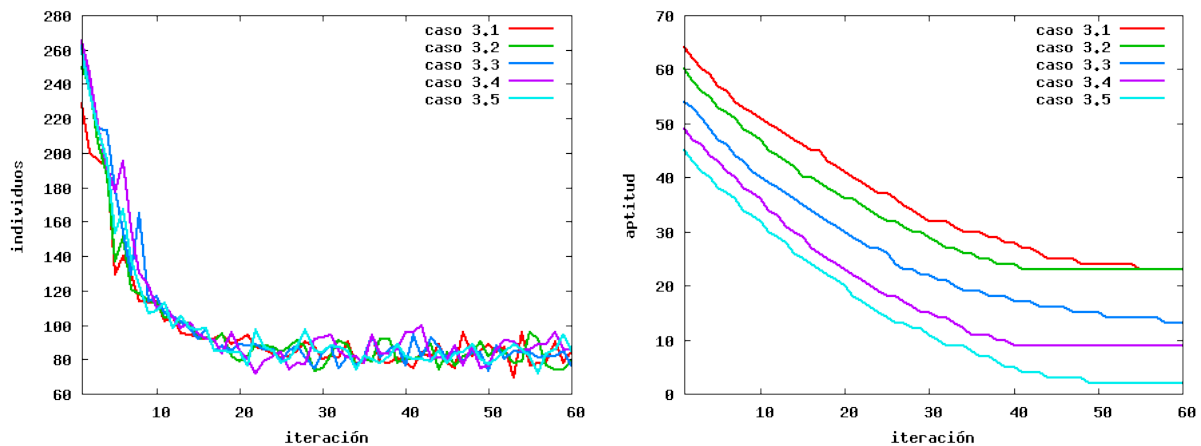


Figura 14: Evolución de la cantidad de individuos mejores a la media y de la aptitud del 20 % más apto de la población en el análisis 3

4. Conclusión Final

De los distintos análisis realizados se pueden deducir, para este problema y algoritmo particular, que:

- La cantidad de individuos de la población tiene incidencia en el resultado pero no es un factor esencial.
- Seleccionar una mayor cantidad de individuos para cruzar genera mayor heterogeneidad en la población.
- Mayor diversidad en la población ayuda a evitar el estancamiento en el progreso hacia una mejor calidad de las soluciones.
- Menos celdas incógnita hacen al problema más resoluble.
- Los más aptos tienden a homogeneizarse en una solución que no suele ser la óptima.

- Si no se logró una solución para la iteración 60, posiblemente nunca se logre.

Estas conclusiones están basadas en las pruebas realizadas y, por tratarse de un algoritmo probabilístico, pueden tener cierto corrimiento.

La última de las conclusiones, referidas a la cantidad de iteraciones donde se estabiliza el progreso en la calidad de la población puede observarse ejemplificada en la figura 15. Este mismo comportamiento se apreció en todos y cada uno de los casos analizados.

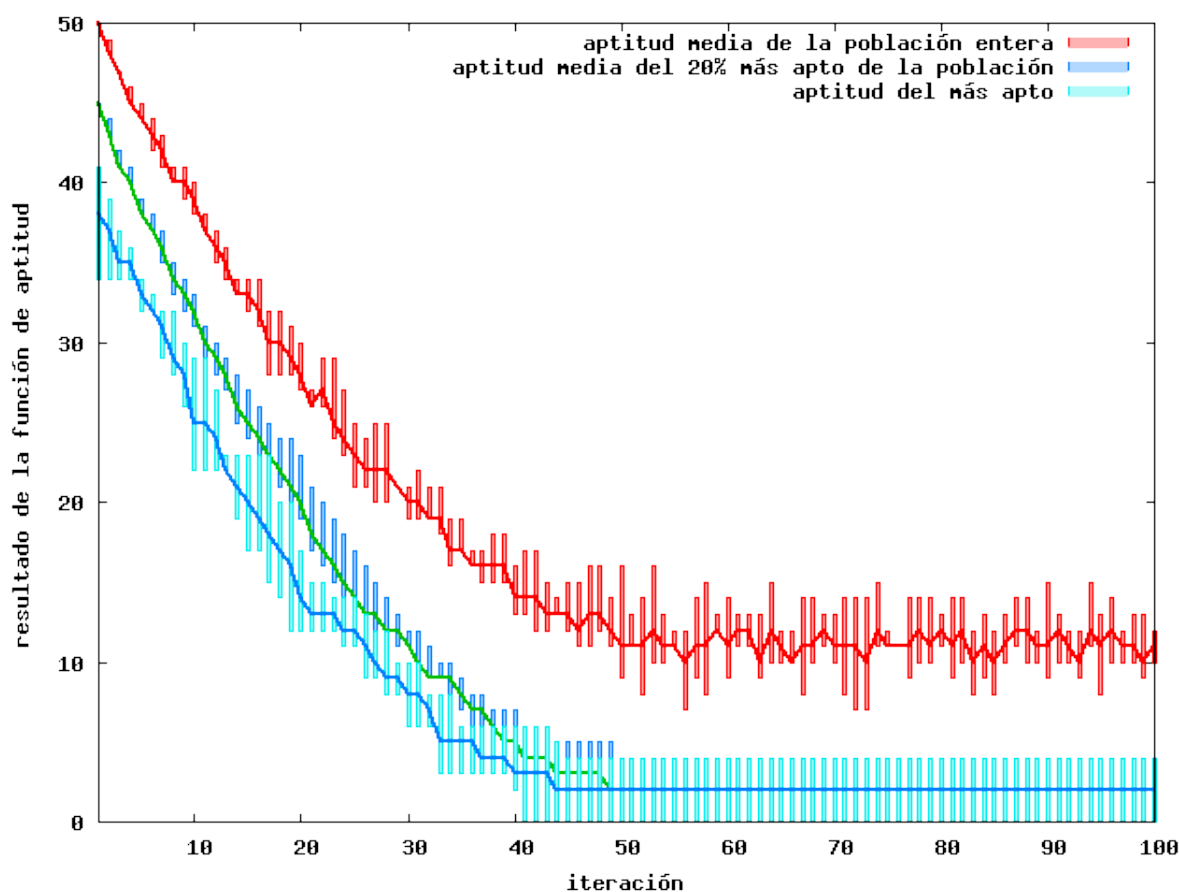


Figura 15: Evoluciones de las aptitudes en el análisis 3.5

Por otro lado, la dificultad presentada en la resolución de este problema tiene relación con la naturaleza del mismo y la forma en la que se busca llegar a su solución. Al tratarse de un asunto determinístico como es la mutua exclusión de las posibilidades en el resultado, es sumamente ineficiente abordarlo desde el punto de vista probabilístico[4]. En el directorio `backtracking/` que acompaña a este documento, hay un programa³ que aborda el problema desde las restricciones y utiliza una técnica llamada *backtracking*. Esta técnica alcanza la solución recorriendo un árbol, volviendo atrás cuando alguna restricción no permite continuar, y logra llegar al sudoku resuelto en fracciones de segundo.

³desarrollado por Ignacio Marambio Catán, estudiante de la Facultad de Ingeniería de la UBA. Adjuntado con su permiso.

Referencias

- [1] Colección Sudoku Mini. *Sudoku, un juego de lógica e ingenio*. Colibrí, 2006.
- [2] SudokuManía.com.ar. Origen del sudoku. Website. <http://www.sudokumania.com.ar/historia-sudoku.htm>.
- [3] Wikipedia. Sudoku. Website. <http://es.wikipedia.org/wiki/sudoku>.
- [4] Wikipedia. Sudoku backtracking. Website. http://es.wikipedia.org/wiki/Sudoku_backtracking.