Tarea 5 - Optimización estocástica

Erick Salvador Alvarez Valencia

CIMAT A.C.,
erick.alvarez@cimat.mx

Resumen En el presente reporte se hablará sobre la implementación de una metaheurística de trayectoria que fue usada en el problema del sudoku. De la misma forma se hablará sobre los resultados obtenidos con instancias de 9x9 y de 16x16 así como una pequeña comparación con el algoritmo de búsqueda local óptima implementado en la tarea pasada.

Keywords: Sudoku, metaheurística de trayectoria, recocido simulado.

1. Implementación de la metaheurística de trayectoria

Para esta tarea se decidió implementar como algoritmo de metaheurística de trayectoria el recocido simulado o somo se conoce en inglés: *Simulated annealing* el cual consiste en el manejo de una temperatura T_i la cual se irá decrementando al pasar el tiempo o las iteraciones. Una vez se tenga la temperatura T_i actualizada se debe iterar sobre un conjunto de vecinos del estado actual del sudoku y por cada vecino que se denotará como $\hat{s}$ se deberá comprobar si el fitness generado por $\hat{s}$ es mejor que el que ya se tiene, es decir, el fitness generado por s donde s es el estado actual. De ser mejor el fitness generado por un vecino se aceptará a dicho vecino como estado actual, en caso contrario se considerará a ese vecino como un movimiento de empeoramiento y se aceptará como nuevo estado actual si al generar un número aleatorio entre cero y uno, dicho número es menor a $e^{\frac{-\Delta E}{T_i}}$ la cual denotaremos como probabilidad de aceptación para un movimiento de empeoramiento, en esta parte ΔE representa la resta del fitness generado por el vecino y el generado por el estado actual. Hay que notar que entre más pequeña sea la temperatura, la probabilidad de aceptación será cada vez menor, y para ello es el uso principal de la temperatura, con ella iremos disminuyendo la probabilidad de aceptación de movimientos de empeoramiento, por lo tanto debemos disminuir a su vez esta temperatura al pasar el tiempo, esta reducción se puede realizar de muchas formas aunque para el presente trabajo sólo se aplicó dos métodos de reducción, el lineal y el logarítmico. En la sección de resultados se mostrarán algunas gráficas que muestran el desempeño del recocido simulado con las reducciones de temperatura como se indicaron anteriormente.
Algo que hay que tomar en cuenta sobre la probabilidad de aceptar un movimiento de empeora es que esta se calcula usando un exponencial y eso es costoso computacionalmente, y por lo tanto se han propuesto algunas otras maneras de calcular esta probabilidad, pese a eso en este trabajo si se usó la manera habitual

de calcular dicha probabilidad. Otra cuestión a considerar es decidir cuántos vecinos se visitarán tras la actualización de la temperatura actual, y de nuevo se comenta que se han propuesto varias formas para la toma de esa decisión pero en el presente trabajo se usó una de las más sencillas y a la vez más utilizadas la cual es decidir un porcentaje de vecinos a visitar, este porcentaje siempre es el mismo tras cada nueva temperatura se tenga.
A continuación se mostrará el pseudocódigo del algoritmo de recocido simulado usado.

Algorithm 1 Recocido simulado.

```
1: procedure SimulatedAnnealing(sudoku, n)
2:     Neighbors ← getNeighbors(sudoku).
3:     ρ ← PORCENTAJE DE VISITA A LOS VECINOS.
4:     Tmax ← TEMPERATURA INICIAL.
5:     Tmin ← TEMPERATURA FINAL.
6:     TimeAct ← 0.0.
7:     TimeMax ← TIEMPO LÍMITE.
8:     Tact ← Tmax.
9:     ESFactor ← ⌈ρ * |Neighbors|⌉
10:    while TimeAct ≤ TimeMax do
11:        randomShuffle(Neighbors).
12:        k ← 0.
13:        moves ← 0.
14:        while moves < ESFactor do
15:            CurrentNeighbor ← Neighbors[k].
16:            FA ← fitness(sudoku).
17:            FP ← fitness(CurrentNeighbor).
18:            ΔE ← FP - FA.
19:            if ΔE ≤ 0 then
20:                sudoku ← CurrentNeighbor.
21:                k ← 0.
22:            else
23:                P ← exp(−ΔE / Tact)
24:                if rand() ≤ P then
25:                    sudoku ← CurrentNeighbor.
26:                    k ← 0.
27:                else
28:                    k ← k + 1.
29:            moves ← moves + 1.
                                  ▷ Actualizar la temperatura linealmente.
30:        TimeAct ← getCurrentTime().
31:        Tact ← Tmax - (Tmax - Tmin) * (TimeAct / TimeMax).
32:    return sudoku.
```

Podemos ver que en el algoritmo anterior que se itera sobre un porcentaje preestablecido de vecinos por la variable ρ, y en caso que un vecino genere un

mejor resultado del que ya se tenía automáticamente nos cambiamos a el, en caso contrario tenemos que ver con cierta probabilidad si aceptamos el movimiento o no, y a la vez como esta probabilidad depende directamente de la temperatura actual esta irá disminuyendo cada vez que la temperatura lo haga.
Podemos ver que la actualización de la temperatura se hace linealmente con respecto al tiempo transcurrido y al tiempo restante, pero como se mencionó anteriormente, esta actualización también se programó de manera logarítmica la cual simplemente sigue la siguiente función:

$$Tact = \frac{Tmax}{log(TimeAct)} \tag{1}$$

2. Ejecuciones y resultados

Para el presente trabajo se realizaron diferentes ditpos de pruebas en las cuales se obtuvieron varios resultados. A continuación se mostrarán los tipos de pruebas realizadas, en las cuales se dirán los parámetros modificados así como los resultados obtenidos. Cabe mencionar que todas las pruebas que se mostrarán fueron realizadas usando el cluster El Insurgente.
Primeramente se trabajó el recocido simulado por iteraciones, este tipo de pruebas se realizaron usando tanto la inicialización aleatoria como la heurística constructiva. Por cada instancia se hicieron 100 ejecuciones del algoritmo en las cuales se usó una semilla diferente y por cada ejecución del recocido simulado se usaron 1000 iteraciones y en cada una el porcentaje de vecinos visitados (estado de equilibrio) fue de 75 %, la temperatura inicial fue de 7 y la final fue de 0.01. A continuación se muestran los resultados obtenidos con la configuración anterior mencionada.

Cuadro 1: Resultados del SA usando 100 ejecuciones y una inicialización aleatoria.

Instancia	Tasa de éxito	Fitness promedio	Fitness mínimo	Fitness máximo	Tiempo promedio (seg).	Tiempo mínimo (seg).	Tiempo máximo (seg).
Easy	84 %	0	0	6	2.14	2.11	2.18
Medium	33 %	2	0	6	2.7	2.65	2.74
Hard	35 %	1	0	5	2.97	2.95	3.01
David Filmer 1	0 %	2	2	6	3.06	3.02	3.22
David Filmer 2	1 %	2	0	4	3.19	3.01	3.31
Inkala	0 %	2	2	5	3.26	3.04	3.30
SD1	1 %	2	0	6	3.07	3	3.19
SD2	2 %	2	0	6	2.98	2.96	3.02
SD3	3 %	2	0	5	3.08	3	3.13
16x16 1	0 %	9	3	15	82.92	10	89.21
16x16 2	0 %	10	2	18	85.23	10	87.69
16x16 3	0 %	10	4	19	84.08	10.05	85.64

Cuadro 2: Resultados del SA usando 100 ejecuciones y una inicialización con heurística constructiva.

Instancia	Tasa de éxito	Fitness promedio	Fitness mínimo	Fitness máximo	Tiempo promedio (seg).	Tiempo mínimo (seg).	Tiempo máximo (seg).
Easy	81 %	0	0	2	2.14	2.08	2.23
Medium	41 %	2	0	7	2.70	2.65	2.93
Hard	19 %	2	0	6	2.96	2.89	3.06
David Filmer 1	0 %	2	2	5	3.07	3.02	3.13
David Filmer 2	1 %	2	0	6	3.17	3.05	3.33
Inkala	0 %	2	2	6	3.26	3.12	3.36
SD1	2 %	2	0	5	3.04	2.98	3.17
SD2	1 %	2	0	6	2.98	2.91	3.07
SD3	4 %	2	0	6	3.07	2.99	3.13
16x16 1	1 %	9	0	16	85.03	10	86.02
16x16 2	0 %	10	2	17	87.65	10	88.71
16x16 3	0 %	11	2	17	86.25	10	87.64

En las dos tablas anteriores podemos apreciar los resultados al aplicar el algoritmo de recocido simulado por iteraciones a todas las instancias. La primer tabla corresponde a los resultados obtenidos con la inicialización aleatoria y la segunda corresponde a la inicialización por heurística constructiva.
Podemos apreciar que ambas tablas proveen resultados muy parecidos en los cuales vemos una alta tasa de éxito para la instancia *Easy* y una tasa media de éxito para las instancias *Medium* y *Hard*, también se puede ver que algunas instancias con grado de dificultad alto como *David Filmer 2*, *SD1*, *SD2* y *SD3* se puedieron resolver en algunas ocasiones. Algo interesante es que en las instancias que no se pudieron resolver tenemos que se pudo obtener un fitness mínimo de 2 errores en las ejecuciones realizadas. Para las instancias de 16x16 tenemos que solamente con la heurística constructiva se pudo resolver una de dichas instancias.
Otra cosa que hay que denotar es que los tiempos de ejecución para las instancias de 9x9 fueron bajos, tenemos que por cada ejecución del algoritmo con los parámetros anteriormente mencionados se tuvo en promedio una tardanza de 3 segundos. En cambio para las instancias grandes de 16x16 esto no fue de la misma forma ya que el promedio de tiempo por ejecución aumentó bastante teniendo ejecuciones de hasta 88 segundos.
Ahora con lo anterior visto se puede realizar una comparativa con los resultados obtenidos en la tarea anterior donde se usó una búsqueda local óptima con la técnica de programación dinámica. A continuación se muestra la misma tabla que se colocó en el informe de la tarea pasada.

Cuadro 3: Resultados de las 100 ejecuciones usando una generación aleatoria y la búsqueda local óptima.

Instancia	Menor Fitness bl. sencilla	Menor Fitness bl. óptima	Tasa de éxito bl. sencilla	Tasa de éxito bl. óptima	Media Fitness bl. sencilla	Media Fitness bl. óptima	Media tiempo de ejecución bl. sencilla (seg)	Media tiempo de ejecución bl. óptima (seg)
Easy	2	0	0 %	28 %	3	22	0.0358684	0.0036767
David Filmer 1	6	2	0 %	0 %	12	7	0.0442347	0.0058113
David Filmer 2	4	2	0 %	0 %	12	7	0.0438622	0.0067342
Inkala	6	2	0 %	0 %	12	7	0.0428983	0.0079581
Medium	5	0	0 %	5 %	18	6	0.0416354	0.0048752
SD1	6	2	0 %	0 %	12	7	0.0441231	0.0061961
SD2	6	2	0 %	0 %	13	7	0.043254	0.0059216
SD3	4	3	0 %	0 %	13	7	0.0429436	0.0059273
Hard	5	0	0 %	3 %	31	6	0.0453188	0.0081378
16x16 1	24	8	0 %	0 %	70	17	1.2585	0.160018
16x16 2	26	10	0 %	0 %	52	21	1.24287	0.203881
16x16 3	27	8	0 %	0 %	59	18	1.27716	0.240503

Si analizamos la Tabla mostrada anteriormente y la comparamos con las dos primeras veremos que hay una gran diferencia con la calidad de las soluciones obtenidas, primeramente se ve que se han resuelto instancias que la búsqueda local no pudo resolver. Otra cosa que notamos es en las instancias que si logró resolver la búsqueda local óptima el recocido simulado generó una mejor tasa de éxitos teniendo por ejemplo en la instancia *Medium* un 5 % con la búsqueda local y un 41 % con la metaheurística de trayectoria. Y debido a que lo anterior es una gran mejora con respecto a la tarea pasada también podemos notar un crecimiento en los tiempos de ejecución con respecto a la búsqueda local y aunque en las instancias de 9x9 no es tanto tiempo en las grandes de 16x16 si tenemos un mayor incremento.

Posteriormente se realizaron pruebas de la metaheurística de trayectoria usando el factor de tiempo en lugar de iteraciones. Esto anterior se usó para la actualización de la temperatura, tal y como se muestra en el pseudocódigo anterior mostrado. En esta parte se hicieron pruebas tanto con inicializaciones aleatorias como inicializaciones con la heurística constructiva. A continuación se muestran las soluciones encontradas para ejecuciones de una hora usando los dos tipos de inicialización y un drececimiento lineal de la temperatura, además con una condición de enfriamiento de aceptación del 80 % de los vecinos, una temperatura inicial de 7 y final de 0.01.

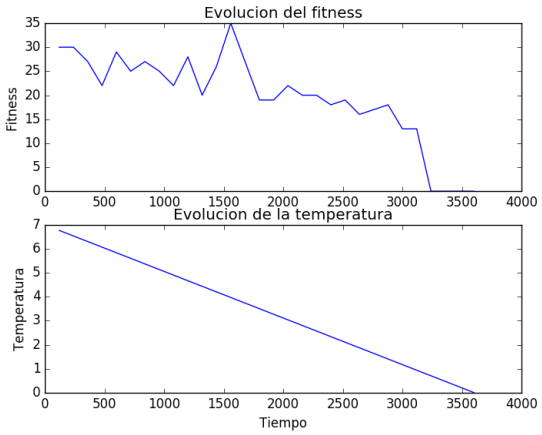
Cuadro 4: Resultados del SA a través de una hora de ejecución y usando una construcción aleatoria.

Instancia	**Fitness inicial**	**Fitness resultante**
Easy	930	0
Medium	727	0
Hard	882	0
David Filmer 1	590	2
David Filmer 2	502	0
Inkala	894	2
SD1	632	2
SD2	692	2
SD3	895	2
16x16 1	2330	6
16x16 2	2210	4
16x16 3	2352	0

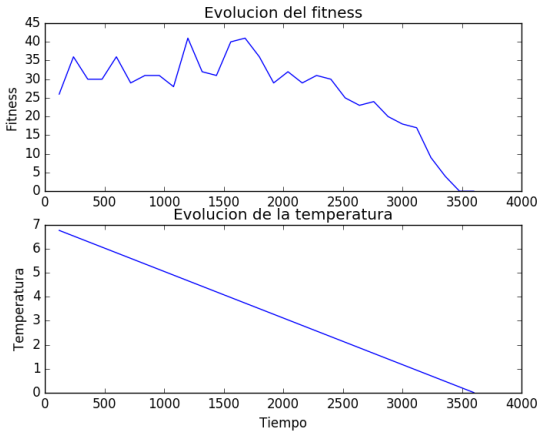
Cuadro 5: Resultados del SA a través de una hora de ejecución y usando una construcción con heurística constructiva.

Instancia	**Fitness inicial**	**Fitness resultante**
Easy	567	0
Medium	561	0
Hard	110	0
David Filmer 1	312	2
David Filmer 2	61	2
Inkala	109	2
SD1	112	0
SD2	212	2
SD3	165	2
16x16 1	1006	6
16x16 2	696	5
16x16 3	840	7

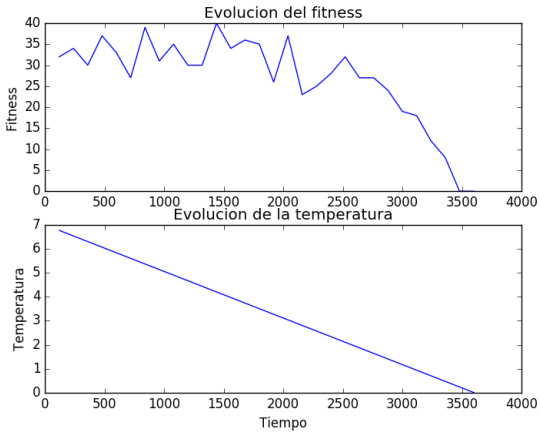
De las dos tablas anteriores podemos ver que los resultados obtenidos obtenidos fueron muy parecidos a los previamente mostrados con la ejecución por iteraciones. Además podemos ver que el tipo de inicialización no influye tanto en los resultados finales ya que pese a que se inicia con menos fitness siempre se cae en los mismos óptimos locales.
Además de los resultados mostrados en los cuadros anteriores también se realizaron gráficas con respecto a la evolución del fitness y la temperatura en el transcurso de la ejecución del algoritmo, las muestras fueron tomadas cada 2 minutos de ejecución. A continuación se mostrarán algunas de las tablas generadas con lo anterior mencionado, no serán las de todas las instancias más sin embargo se mostrarán las más relevantes.



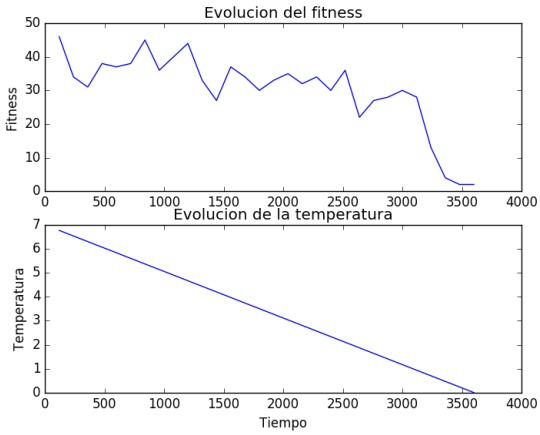
(a) Figura 1. Evolución de la temperatura y el fitness de la instancia *Easy*.



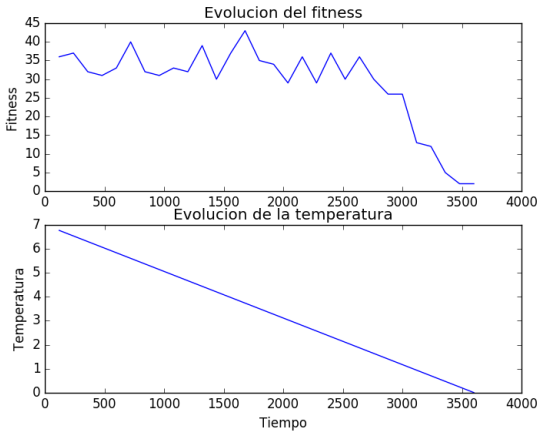
(b) Figura 2. Evolución de la temperatura y el fitness de la instancia *Medium*.



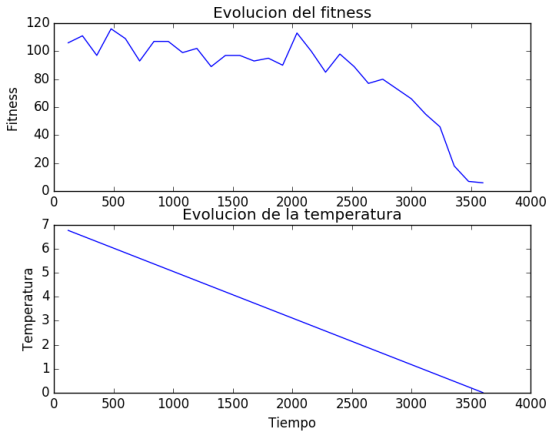
(c) Figura 3. Evolución de la temperatura y el fitness de la instancia *Hard*.



(d) Figura 4. Evolución de la temperatura y el fitness de la instancia *David Filmer* 1.



(e) Figura 5. Evolución de la temperatura y el fitness de la instancia $SD1$.



(f) Figura 6. Evolución de la temperatura y el fitness de la instancia $16x16$ 1.

En las 6 Figuras anteriores podemos apreciar la evolución tanto de la temperatura como del fitness a lo largo de la hora de ejecución que se tuvo del algoritmo. Lo que podemos notar al principio es que como se mencionó anteriormente la temperatura fue disminuyendo linealmente y esto se aprecia de manera muy clara en todas las gráficas. Por otra parte tenemos que la evolución del fitness a lo largo de la ejecución presenta un descenso no lineal, se puede apreciar que en ciertas partes el fitness incluso aumenta en lugar de descender y ahí es

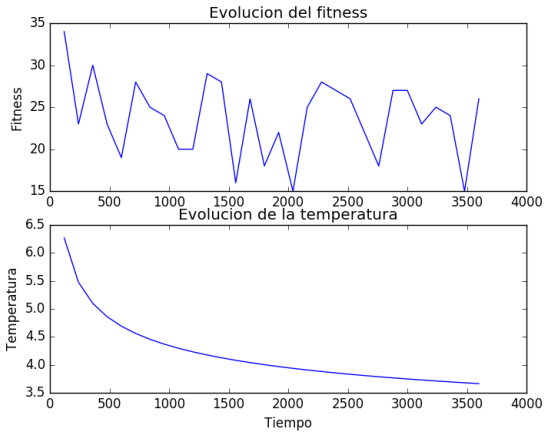
cuando vemos que se están aceptando movimientos de empeoramiento, esto anterior generalmente ocurre en atapas tempranas a medias de la ejecución, después de la mitad del tiempo de la misma esto ya no ocurre tan segudio debido a que la probabilidad de aceptación va decreciendo en conjunto con la temperatura.
Podemos ver que hay instancias como David Filmer 1, SD1 y 16x16 1 donde se encuentran óptimos locales pasado el segundo 3000 y en esa parte es donde se queda el algoritmo haciendo una búsqueda intensiva.

De la misma forma se realizaron ejecuciones de una hora pero se cambió el modo en que la temperatura se actualizaba, ahora se generó un descenso logarítmico en el cual la temperatura desciende muy rápido en etapas tempranas de la ejecución pero en etapas medias y tardías el descenso es mucho más lento.
A continuación se muestran los resultados al aplicar esta actualización de temperatura. Para este caso se usó una temperatura inicial de 30 ya que debido al logaritmo esta necesita ser más grande que la usada en el descenso lineal.

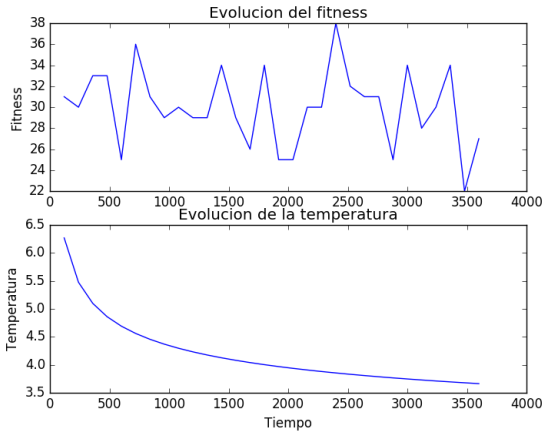
Cuadro 6: Resultados del SA a través de una hora de ejecución y usando un descenso logarítmo para la temperatura.

Instancia	**Fitness inicial**	**Fitness resultante**
Easy	373	26
Medium	66	27
Hard	217	31
David Filmer 1	163	31
David Filmer 2	108	34
Inkala	211	30
SD1	258	34
SD2	210	29
SD3	314	32
16x16 1	1240	107
16x16 2	941	109
16x16 3	643	108

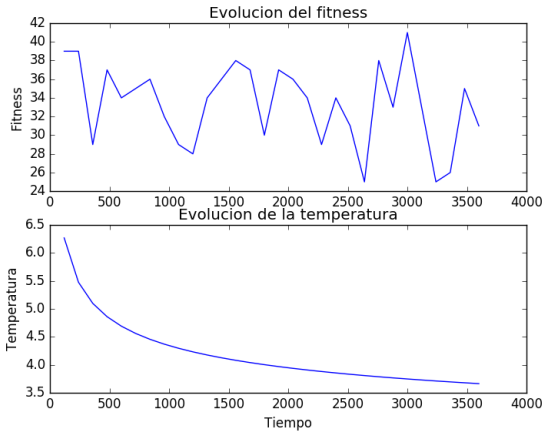
Lo primero que podemos ver en la tabla anterior es el hecho de que los resultados obtenidos fueron mucho peores que los obtenidos con el descenso lineal, se aprecia que ni siquiera se pudieron resolver las instancias fáciles.
A continuación se muestran unas pocas gráficas sobre la evolución del fitness y la temperatura a través de la ejecución del algoritmo.



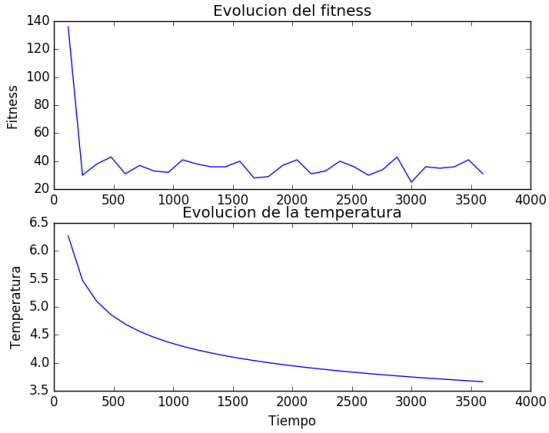
(g) Figura 7. Evolución de la temperatura y el fitness de la instancia *Easy* con descenso logarítmico.



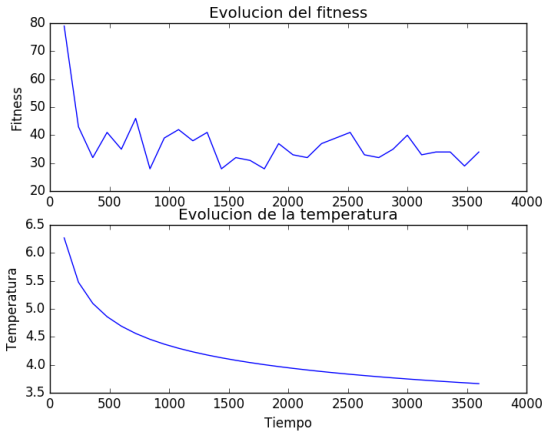
(h) Figura 8. Evolución de la temperatura y el fitness de la instancia *Medium* con descenso logarítmico.



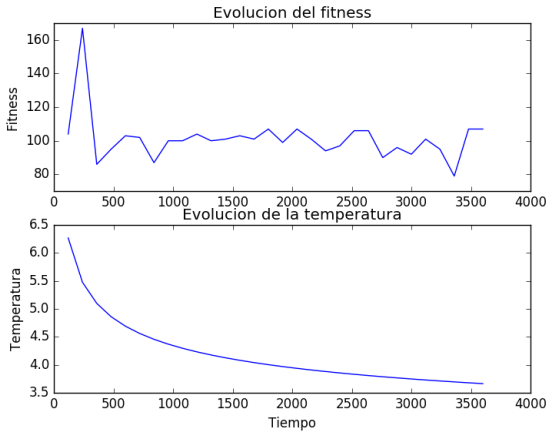
(i) Figura 9. Evolución de la temperatura y el fitness de la instancia *Hard* con descenso logarítmico.



(j) Figura 10. Evolución de la temperatura y el fitness de la instancia *David Filmer* 1 con descenso logarítmico.



(k) Figura 11. Evolución de la temperatura y el fitness de la instancia $SD1$ con descenso logarítmico.



(l) Figura 12. Evolución de la temperatura y el fitness de la instancia $16x16$ 1 con descenso logarítmico.

En las Figuras anteriores se muestran tanto la evolución del fitness como la temperatura usando un desenso logarítmico, en donde se aprecia claramente que la temperatura desciende con respecto a la función logaritmo, aquí el descenso se realiza muy rápido en el primer tercio de la ejecución del algoritmo, de ahí el descenso se vuelve más lento. Se puede apreciar que usando una temperatura inicial de 30 se tiene que el rango de la actualización de la temperatura a lo largo de la hora de ejecución es de 6 a 4, un rango muy pequeño y probablemente esto

fue la causa de que los resultados del algoritmo no fueran tan buenos como con el descenso lineal.
En la parte de la actualización del fitness se aprecia en algunas gráficas un descenso muy irregular ya que este a lo largo de casi todo el tiempo tiene mucha subidas y bajadas, a diferencia del SA con descenso lineal el cual las subidas se tenían en la parte inicial de la ejecución, aquí se tiene en la mayor parte. Incluso en instancias como *Hard* tenemos que no se llegó a encontrar un óptimo local como tal ya que la gráfica indica que el fitness estaba dando altibajos en las últimas partes de la ejecución. En la instancia *David Filmer* 1 podemos notar que casi desde el comienzo de la ejecución el fitness descendió muy rápido pero se estancó en los valores de 35-40.

Notas y comentarios finales. Como conclusión se tiene que la implementación de la metaheurística de trayectoria resultó exitosa ya que se pudieron obtener mejores resultados que los arrojados por la búsqueda local óptima implementada en la tarea pasada, en los cuales se pudo resolver más instancias de los sudokus además las que ya se tenía solución se consiguió una tasa de éxito mucho mayor. Se pudo apreciar los diferentes tipos de configuraciones realizadas al recocido simulado, desde la implementación que usaba como criterio de avance las iteraciones hasta la que usaba el tiempo (ejecuciones de 30 min. y una hora), en donde se obtubieron resultados muy similares. De la misma forma se hizo una implementación en donde el descenso de la temperatura era de manera logarítmica y en este caso los resultados obtenidos tras ejecuciones de 1 hora no fueron buenos esto debido a la función logaritmo ya que este crece/decrece (según sea el caso) muy lentamente y aquí se necesitarían realizar ejecuciones de mucho más tiempo para obtener un rango de temperaturas más amplio.
Como nota final, no se incluyeron todas las gráficas en el presente informe debido a que se generó un conjunto muy grande de ellas al terminar todas las pruebas, sin embargo todas se incluirán en los archivos que se adjuntarán con el informe, así como los archivos de texto de los resultados arrojados por el algoritmo.