# Tarea 6 - Optimización estocástica

Erick Salvador Alvarez Valencia, *CIMAT A.C.*

***Index Terms*—Metaheurísticas de trayectoria, algoritmo memético, multi dyn.**

## I. Introducción

En el presente reporte se describirá la implementación de un algoritmo memético con un enfoque multi-objetivo el cual se encarga de resolver problemas mono-objetivo. Dicho algoritmo fue usado para tratar el problema de resolución de sudokus, en donde se probó con varias instancias de nivel: fácil, medio y difícil.
Se decribirá en términos generales el modo en cómo se implementó dicho algoritmo así como los resultados obtenidos en cada instancia y un pequeño análisis de diversidad de las ejecuciones realizadas.

mds
Abril 10, 2018

## II. Algoritmo memético

Para esta tarea se implementó una metaheurística poblacional enfocada a resolver el problema del sudoku, la cual se compone de un algoritmo memético con una modificación que se enfoca en la construcción de la población basado en su valor de fitness y su diversidad.
Lo primero a tener en cuenta es que un algoritmo memético se diferencia de un genético en el hecho de que además de hacer una mutación a los individuos se realiza una búsqueda local en los mismos, esto para acelerar el proceso de convergencia hacia alguna solución. De la misma forma existen implementaciones en las que se suprime el proceso de mutación, pero en el caso de la presente implementación eso no ocurrió.
La implementación comenzó generando una población de $N$ individuos donde $N$ es un valor arbitrario, cada individuo de dicha población es una instancia de un sudoku la cual se generó como se ha manejado en las tareas pasadas, se guardan los números de las permutaciones por cada bloque, en donde dichos números son los que no vienen por defecto en la instancia. De igual manera la función de fitness para una instancia del sudoku permaneció igual a como se tenía implementada en las tareas pasadas. La penalización viene dada por la suma de los errores en las filas y columnas del sudoku, y si un error es generado por un conflicto con un número que ya estaba por defecto en la instancia, la penalización sumará 100 por ese error.
Ahora, se describirán los operadores de cruza y mutación usados en la implementación del algoritmo memético.

Para la parte de la cruza el operador generó a los hijos copiando la información de los bloques enteros de los padres con cierta probabilidad, es decir, por cada bloque de números que tiene la instancia de un padre se copian dichos bloques a uno u a otro hijo dependiendo el valor de una probabilidad $P$, dicha probabilidad fue elegida uniformemente. A continuación se muestra el algoritmo usado para la cruza de individuos.

**Algorithm 1** Cruza de individuos.

```
1:  procedure CROSSOVER(p1, p2)
2:      sz ← p1.size().        ▷ Obtener el número de bloques.
3:      son1 ← [].
4:      son2 ← [].
5:      for i = 1 → sz do
6:          p ← random(0, 1).
7:          if p < 0.5 then
8:              son1 ← son1 ⋃ p1[i].
9:              son2 ← son1 ⋃ p2[i].
10:         else
11:             son1 ← son1 ⋃ p2[i].
12:             son2 ← son1 ⋃ p1[i].
13:     return son1, son2.
```

Para el operador de mutación, se fue recorriendo cada bloque de la instancia y por cada número de el i-ésimo bloque, se permutaba con otro número en el mismo bloque elegido al azar si es que la probabilidad de mutación se cumplía, hay que recordar que dicha probabilidad debe ser bastante baja. A continuación se muestra el pseudocódigo del operador de mutación.

**Algorithm 2** Mutación de individuos.

```
1:  procedure MUTATION(ind, mutationRate)
2:      sz ← ind.size().        ▷ Obtener el número de bloques.
3:      for i = 1 → sz do
4:          for j = 1 → ind[i].size() do        ▷ Iterar en los
    números del bloque.
5:              if random(0, 1) ≤ mutationRate then
6:                  flip ← randomSelect(ind[i]). ▷ Seleccionar
    un individuo aleatoriamente del mismo bloque.
7:                  swap(ind[i][j], ind[i][flip]).
8:      return ind.
```

Para esta parte se puede pensar que hay la posibilidad de que al permutar dos elementos de un bloque, la mutación vuelva a hacer la permutación de dichos elementos cuando se visite el otro elemento. Hay que tener en cuenta como se comentaba anteriormente que la probabilidad de mutación se debe escoger muy baja para que en teoría solo permute un elemento o dos por individuo, la probabilidad tomada para las pruebas se mostrará en la siguiente sección.

Finalmente se describirá el enfoque basado en diversidad que fue implementado para poder obtener mejores resultados en el algoritmo memético. Este enfoque hace una optimización multiobjetivo en donde los individuos seleccionados son aquellos que proveen buenos fitness y una diversidad alta, esto anterior servirá para evitar convergencias prematuras dado el hecho de que nuestra población se empieze a asemejar y caiga en un óptimo local que no tiene tan buenos resultados. Además esta preservación de individuos con gran diversidad biene controlada por un factor que va decreciendo a lo largo del tiempo de la ejecución, dicho factor se encarga de penalizar a individuos con baja diversidad para que estos no sean elegidos en la parte multi-objetivo. A continuación se muestra el esquema general para el algoritmo memético con estrategia de selección por diversidad llamada multi-dyn.

**Algorithm 3** Selección de individuos usando estrategia de diversidad.

1: **procedure** MULTIDYN($currentMembers$, $n$)
2: newPop ← .
3: sortByFitness(currentMembers).
4: newPop ← newPop ⋃ currentMembers[0].
5: currentMembers ← currentMembers - currentMembers[0].
6: **while** |newPop| $< n$ **do**
7: DCN ← getDCN(currentMembers, newPop).
8: D ← DI - DI * $\frac{currentTime}{maxTime}$.
9: sortAsc(DCN).
10: nonDominated ← {}.
11: fitnessMax ← $\infty$.
12: **for** i = 1 → DCN.size() **do**
13: **if** DCN[i].diversity ≥ D **then**
14: **if** currentMembers[i].fitness **then**
15: nonDominated ← nonDominated ⋃ DCN[i].id.
16: fitnessMax ← currentMembers[DCN[i].id].fitness.
17: idx ← randomVal(nonDominated).
18: newPop ← newPop ⋃ currentMembers[idx].
19: currentMembers ← currentMembers - currentMembers[idx].
20: **return** $newPop$.

El algoritmo anterior muestra el esquema de selección de individuos basados en diversidad, primeramente hay que comentar que $currentMembers$ es una población constituida por la población en un tiempo $t$ unida a otra población de descendencia construida en base a la población en el tiempo $t$. Lo primero que se hace es preservar al individuo que da el mejor fitness y quitarlo de $currentMembers$. Posteriormente para añadir un miembro a la nueva población se obtiene el DCN de la población de $currentMembers$, dicho DCN no es más que el valor de mínima distancia que genera un individuo de $currentMembers$ con respecto a los individuos de la nueva población, dicha distancia puede manejarse de varias formas, para este trabajo se usó la distancia de Hamming. Estos valores nos miden la diversidad que generan los individuos con respecto a los ya considerados en la nueva población, esto es muy importante ya que este será un factor a tener en cuenta para escoger nuevos individuos. El siguiente paso será la parte multi-objetivo ya que se buscan individuos que tengan un buen fitness y a la vez una buena diversidad, y para ello se buscan en la población $currentMembers$ los individuos no dominados con respecto a estas dos características, es decir, los que se encuentren en el frente de pareto del conjunto. Una vez que se tiene calculado dicho frente, simplemente se elige un individuo al azar, se añade a la nueva población y se elimina del conjunto $currentMembers$.
Otra cosa a tener en cuenta es el hecho de que además de lo anterior comentado, se está imponiendo una penalización, la cual es medida por la variable $D$. Dicha penalización ignora a los individuos que tengan una diversidad menor a dicho $D$, pero también esta variable se va reduciendo linealmente con el tiempo transcurrido de la ejecución, esto es muy importante ya que en generaciones tempranas este valor tiende a ser alto y por lo cual ignora muchos individuos que tengan diversidad muy pequeña, pero al transcurrir el tiempo, esto se va reduciendo y deja pasar individuos más pobres en diversidad. Esto anterior sirve para hacer una búsqueda más exploratoria en etapas tempranas de la ejecución y una búsqueda más intensiva en etapas tardías, de esta forma es más fácil encontrar el óptimo global de la función.
A continuación se mostrará la función principal del algoritmo memético, en donde se incluye la implementación anterior.

**Algorithm 4** Algoritmo memético aplicado a la resolución de sudokus.

1: **procedure** MEMETICALGORITHM($n$)
2: pop ← randomInitialization(n).
3: localSearch(pop).
4: **while** Not convergence **do**
5: selected ← Evaluation(pop).
6: offspring ← MatingSelection(selected). ▷ Generar una población de hijos mediante un torneo binario, cruza y mutación a la población actual.
7: offspring ← LocalSearch(offspring).
8: pop ← MultiDyn(pop ⋃ offspring).
9: **return** $pop$.

Podemos ver que el algoritmo anterior parte del concepto de un algoritmo memético, ya que se se genera una población de descendencia usando los operadores de cruza y mutación, además se aplica una búsqueda local a dichos individuos para acelerar la convergencia, la parte que cambia es la generación de la nueva población, en donde se aplica el enfoque multi-dynamic descrito anteriormente.

## III. EJECUCIONES Y RESULTADOS

Para el presente algoritmo se realizaron 30 ejecuciones por instancia, las cuales fueron hechas en el cluster de CIMAT, además de que cada ejecución se realizó con los mismos parámetros los cuales son descritos a continuación.

1. **Tiempo por ejecución:** 30 minutos.

2. **Tamaño de la población:** 100 individuos y cada generación de hijos era de 100 individuos también.
3. **Probabilidad de cruza:** 80 %.
4. **Probabilidad de mutación:** 1 %.
5. **Valor de DI:** 20.
6. **Tamaño del torneo:** 2 individuos (binario).
7. **Búsqueda local usada:** Programación dinámica.

A continuación se muestra una tabla con los resultados obtenidos de las 30 ejecuciones por instancia.

Cuadro I: Resultados del algoritmo poblacional mediante 30 ejecuciones.

| Instancia | Tasa de éxito | Fitness promedio | Fitness mínimo | Fitness máximo | Tiempo promedio (seg). |
|---|---|---|---|---|---|
| **Easy** | 100 % | 0 | 0 | 0 | 4.45 |
| **Medium** | 100 % | 0 | 0 | 0 | 5.23 |
| **Hard** | 100 % | 0 | 0 | 0 | 5.84 |
| **David Filmer 1** | 63 % | 2 | 0 | 2 | 1654.31 |
| **David Filmer 2** | 20 % | 2 | 0 | 2 | 1692.4 |
| **Inkala** | 100 % | 0 | 0 | 0 | 472.2 |
| **SD1** | 100 % | 0 | 0 | 0 | 68.75 |
| **SD2** | 100 % | 0 | 0 | 0 | 72.49 |
| **SD3** | 100 % | 0 | 0 | 0 | 65.12 |

Podemos ver en el Cuadro 1. que todas las instancias de sudoku fueron resueltas y casi todas con el 100 % de tasa de éxito excepto para las instancias de David Filmer 1 y 2 las cualesson consideradas como las instancias más difíciles del conjunto y en especial David Filmer 2, para la cual sólamente se obtuvo una tasa de éxito de un 20 %. Además se puede apreciar que para las tres primeras instancias las cuales son consideradas como fáciles, el promedio de tiempo para la resolución de las mismas fue bastante bajo, en cambio para las instancias de David FIlmer 1 y 2 se requirió casi todo el tiempo de la búsqueda para resolverlas.
Ahora se mostrará una tabla en donde se muestran los resultados obtenidos en la tarea 5 donde se aplicó una metaheurística de trayectoria y se compararán los resultados obtenidos con la actual.

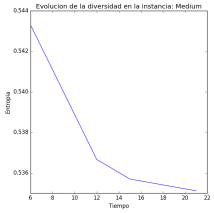
Cuadro II: Resultados del SA usando 100 ejecuciones y una inicialización con heurística constructiva.

| 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 |

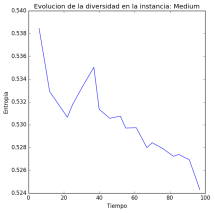
Primero hay que comentar que para esta tarea no fueron probadas las instancias de 16x16. Posteriormente podemos ver que en el Cuadro 2 no se pudieron resolver todas las instancias empleando la metaheurística de trayectoria la cual fue un recocido simulado, y para las que se pudieron resolver como Easy, Medium, Hard, DF2 no se obtuvo tasas de éxito tan promisorias como las obtenidas empleando el algoritmo actual.

### *III-A. Evaluación de la diversidad*

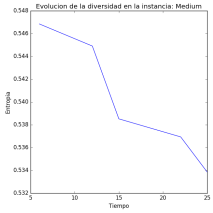
A continuación se mostrarán algunas gráficas obtenidas realizando la evaluación de la diversidad del algoritmo mediante ciertas ejecuciones. Hay que comentar que no se obtuvieron las gráficas de evaluación de diversidad promedio para las 30 ejecuciones debido a que se capturaron datos cada 5 segundos y se dió como forma de parada al algoritmo cuando llegara a la mejor solución, la cual equivale al fitness 0. Por lo anterior no se capturaron los datos suficientes para poder generar gráficas de promedio correctas.




(a) Medium 1

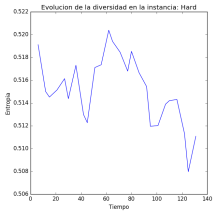



(b) Medium 13



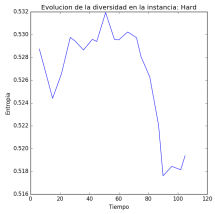

(c) Medium 25

Figura 1: Evolución de la diversidad para la instancia Medium.



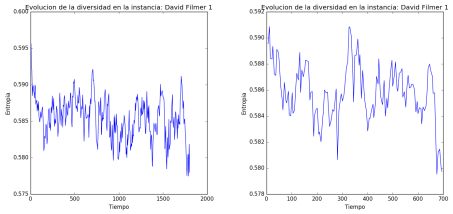

(a) Hard 3


(b) Hard 19




(c) Hard 22

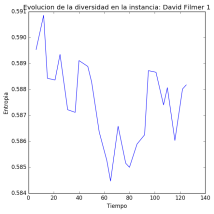
Figura 2: Evolución de la diversidad para la instancia Hard.




(a) DF1 1

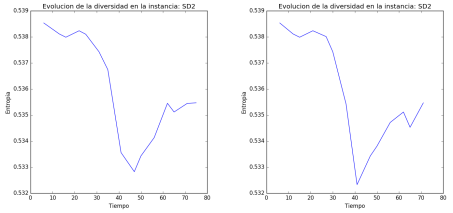



(b) DF1 9




(c) DF1 11

Figura 3: Evolución de la diversidad para la instancia David Filmer 1.




(a) SD2 2




(b) SD2 5


(c) SD2 8

Figura 4: Evolución de la diversidad para la instancia SD2.

En las Figuras anteriores podemos apreciar la evolución de la diversidad de 3 corridas distintas de la misma instancia (por Figura), dichas corridas fueron seleccionadas aleatoriamente para generar las gráficas mostradas. Podemos apreciar que en la mayoría de ellas se presentan saltos en la diversidad, y esto es debido a la selección de los individuos. En la instancia de David Filmer 1 es donde se presentan más saltos y es donde vemos que hubo más exploración por parte del algoritmo para encontrar los óptimos locales.
Otra característica interesante que podemos notar es que a través de casi toda la ejecución no se aprecia un gran decrecimiento de la diversidad, y esto es gracias a la parte de selección por multi dynamic ya que esta siempre se encarga de preservar individuos con fitness muy bajos, pero con diversidad alta, incluso, dichos individuos pueden contener sólo dos errores pero seguirán conteniendo un alto grado de diversidad.

## IV. Conclusión

En el presente reporte se mostró la implementación de un algoritmo memético con un enfoque de selección de individuos basado en optimización multi-objetivo que cuida tanto el fitness como la diversidad de los individuos, dicho algoritmo se probó en el problema del sudoku obteniendo muy buenos resultados con un conjunto de instancias de nivel fácil, medio y difícil, en las cuales se pudo lograr resolver todas aunque en las difícil no se logró obtener una tasa de exito de un 100 %. De la misma forma se hizo un análisi de diversidad de ciertas corridas del algoritmo con unas cuantas instancias y se pudo notar que la diversidad de los individuos es alta a través de toda la ejecución y esta no decrece tan rápido impidiendo una convergencia prematura hacia óptimos locales.
Como comentario final, el programa se probó usando la búsqueda local por escalada, y aunque con esta también se podía resolver varias instancias, se tuvo muy malos resultados para resolver las instancias de David FIlmer 1 y 2. Como trabajo futuro se podría hacer una verificación de lo sucedido y hacer las debidas correcciones.

## Referencias

[1] C. Segura, S. Peña, S. Botello and A. Hernandez, *The Importance of Diversity in the Application of Evolutionary Algorithms to the Sudoku Problem*, CIMAT A.C, 2016, IEEE.