Máster Universitario en Ingeniería Informática

Inteligencia Computacional

Práctica de algoritmos evolutivos

Problemas de optimización combinatoria
QAP

UNIVERSIDAD
DE GRANADA

Carlos Morales Aguilera

10 de Enero de 2020

Índice

1. Introducción

El problema de asignación cuadrática *QAP* es un problema básico de optimización combinatoria con muchas aplicaciones. *QAP* es un problema estándar en la teoría de localización. En este se trata de asignar N unidades a N localizaciones en donde se considera un coste asociado a cada una de las asignaciones. Este coste depende de dos factores principales: distancias y flujos entre unidades.

Al tratarse de un problema de optimización, se trata de minimizar el coste descrito. Se consideran por lo tanto como entrada dos matrices $F = (f\{i,j\})$ y $D = (d\{k,l\})$ donde $f\{i,j\}$ representa el flujo entre las unidades i y j, y $d\{k,l\}$ representa la distancia entre las localizaciones k y l. El problema por lo tanto se modela de la siguiente forma:

$$
\begin{aligned}
\min \quad & \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} f_{ij} d_{kp} x_{ij} x_{kp} \\
\text{s.a.} \quad & \sum_{i=1}^{n} x_{ij} = 1 && 1 \le j \le n\,, \\
& \sum_{j=1}^{n} x_{ij} = 1 && 1 \le i \le n\,, \\
& x_{ij} \in \{0,1\} && 1 \le i, j \le n\,.
\end{aligned}
$$

Por lo tanto, el objetivo deseado es encontrar mediante el uso de algoritmos evolutivos una buena aproximación del problema propuesta, conociendo los mejores casos existentes actualmente. Para ello se plantearán diferentes variantes de un algoritmo genético y se analizarán los resultados obtenidos.

Finalmente, tras observar los resultados obtenidos, se alcanzarán una serie de conclusiones acerca de las aplicaciones de algoritmos evolutivos para este tipo de problema y sobre los mejores resultados obtenidos en la práctica realizada.

2. Datos utilizados - QAPLIB

Los casos de prueba propuestos se encuentran disponibles en la biblioteca *QAPLIB*. Específicamente se va a tratar con el fichero *tai256c.dat*, el cual posee un tamaño de problema de 256 elementos, lo cual significa que se obtendrán permutaciones de 256 elementos.

2.1. Representación del problema

El problema anteriormente indicado se representa mediante su tamaño y matrices asociadas en el fichero de la siguiente forma:
n
A
B

donde n representa el tamaño de las permutaciones del problema, A y B son matrices, de flujos y distancias, indistintamente, ya que el problema es simétrico.

2.1.1. Representación de la solución

Las soluciones representan las diferentes asignaciones mediante un vector que representa la permutación de los diferentes elementos. Esta representación de la solución poseerá por lo tanto dos elementos principales:

- **Vector de permutación**: Que representa las asignaciones de las unidades a las localizaciones, siendo la posición del vector la unidad y el contenido de dicha posición la localización.
- **Coste asociado**: Representa el coste obtenido de la permutación.

2.1.2. Representación de distancias y flujos

En los ficheros proporcionados se encuentra la información asociada a las distancias y flujos entre unidades y localizaciones, la cual debe ser almacenada y procesada en el problema, para lo cual se ha creado una estructura de datos. Dicha estructura de datos poseerá dos matrices correspondientes a los flujos y distancias, de tal forma que una posición *f{i,j}* representará el flujo entre la unidad u_i y la unidad u_j. Igualmente se actuará y representará con la matriz de distancias.

El problema posee matrices cuadradas por lo que la representación de las mismas consistirá en almacenar dos matrices cuadradas del orden del tamaño del problema, siendo este el primer parámetro proporcionado en el fichero.

2.1.3. Función de fitness

La función de *fitness* u objetivo es aquella que nos permite determinar el valor real de las asignaciones que se realizan en el problema, de la cual se ha de obtener los resultados mínimos posibles. Relacionada con la introducción del problema, se define la función objetivo de la forma:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij} \cdot d_{S(i)S(j)}$$

Teniendo en cuenta dicha función, el objetivo será disminuir el coste asociado a una permutación, obtenido por la aplicación de dicha función sobre los elementos, todo esto dentro de un tiempo de ejecución razonable.

3. Implementación

3.1. Justificación del código

El código original de la práctica es una implementación propia realizada durante las prácticas de la asignatura de *Metaheurísticas* del Grado en Ingeniería Informática en la *UGR* en el año 2017.

La implementación ha sido modificada y adaptada a los nuevos requisitos de esta práctica y por lo tanto los resultados variarán en los nuevos modelos. Además se han realizado algunas correcciones y mejoras.

3.2. Algoritmos genéticos

La implementación realizada parte de dos enfoques diferentes de algoritmos genéticos, basados en la evolución de una población. Estos enfoques son los de *Generacional* y *Estacionario*. Aunque los enfoques varíen, los operadores serán idénticos y variará la utilización de los mismos. Adicionalmente, cabe destacar que los resultados deberán ser diferentes debido a los diferentes enfoques.

Por otro lado, es importante definir los aspectos comunes en ambos enfoques considerados en las implementaciones de los mismos:

- **Primera población**: Esta se generará de forma aleatoria, por lo que se añade un factor aleatorio a las posibles soluciones, el cual se intentará paliar sus efectos.
- **Transposiciones**: Las transposiciones consideradas durante todo el algoritmo, tanto en los propios algoritmos genéticos como en las búsquedas locales que se realicen serán del tipo **2-opt**, es decir, intercambio de dos elementos de una solución.
- **Condición de parada del algoritmo**: Se ha establecido como máximo un número de iteraciones, el cual considera el número de llamadas a la función objetivo como una iteración. El límite establecido ha sido de 50.000 iteraciones.

Por lo tanto, el procedimiento común entre los dos enfoques será el siguiente:

1. **Selección**: Se escogen los individuos de la población que formarán parte del proceso de cruce que de lugar a la siguiente generación. Estos individuos, denominados *padres*, serán escogidos mediante un procedimiento de **Torneo Binario**, y serán utilizados para los diferentes cruces que se realicen.
2. **Cruce**: Se utilizan los *padres* escogidos en el proceso de selección para determinar nuevos individuos de la población, denominados *hijos*, los cuales formarán parte de la siguiente generación. En esta práctica se utilizarán diferentes operadores de cruce como son **Cruce por posición** o **Cruce OX**.
3. **Mutación**: Se utilizará una función de probabilidad con el fin de determinar el número de mutaciones realizadas. Estas mutaciones consistirán en intercambiar genes al azar de un individuo.
4. **Elitismo**: Con el fin de mejorar la población en la mayor medida posible, se escogerán los individuos con menor coste, descartando aquellas con mayor coste, es decir, descartando los **peores individuos**.

3.2.1. Algoritmo genético generacional

Basado en las generaciones, durante cada una de las iteraciones del algoritmo el enfoque consiste en generar una nueva población completa con nuevos individuos, manteniendo los mejores individuos de la anterior población, de forma que se mantengan los mejores resultados obtenidos, y se añade una mayor diversidad.

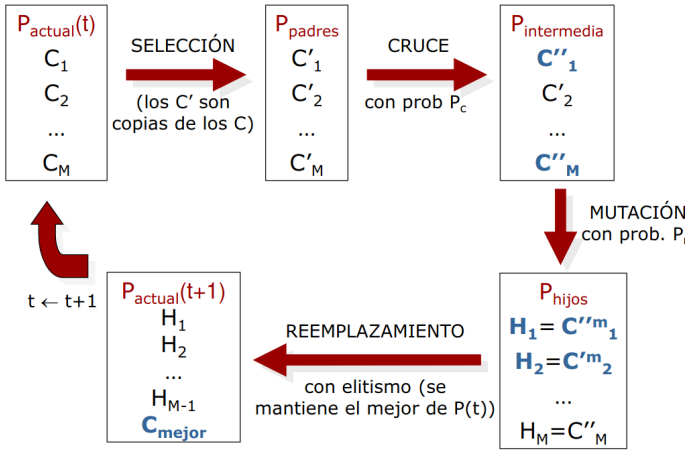


Imagen 1: Procedimiento enfoque generacional [BIO]

3.2.2. Algoritmo genético estacionario

A diferencia del enfoque previamente explicado, este algoritmo busca mantener el estado de la población y enfocarse en la mejora de determinados individuos, por lo que en vez de generar toda una nueva población, escoge una cantidad determinada de padres de la población y se aplican los operadores, sustituyendo estos hijos por los peores individuos de la población, si estos son peores que los nuevos hijos obtenidos.

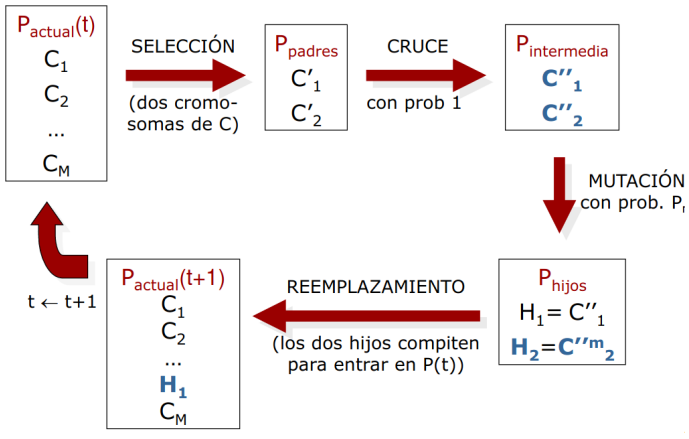


Imagen 2: Procedimiento enfoque estacionario [BIO]

3.2.3. Operadores

Operador de selección: Consistente en un **Torneo Binario**, el cual escoge aleatoriamente dos individuos de la población, comparando sus costes asociados y devolviendo el que posea menor coste entre ellos. Para ello se debe cumplir la restricción de que los individuos escogidos no sean el mismo. El objetivo final de la aleatoriedad del mismo es poder explorar individuos que una primera instancia no son tan prometedores, y que los padres obtenidos finalmente tiendan a parecerse.

Operadores de cruce:

- **Cruce por posición**: Se heredan los elementos comunes de ambos padres en el hijo, y los restantes se rellenan de forma aleatoria, por lo que los hijos mantienen las características comunes y exploran en aquellas que se encuentra cierta diversidad.
- **Cruce por OX**: Escoge una parte de los elementos de uno de los padres para rellenar los primeros elementos del hijo, y se rellenan los restantes de forma iterativa con los elementos restantes del segundo padre.

Operador de mutación: Este operador consiste en la generación de vecinos mediante el intercambio de dos índices del vector de permutación.

Operador de reemplazo: Consiste en sustituir los peores elementos de la población por los nuevos individuos siempre que estos mejoren a dichos individuos propuestos para su descarte.

3.3. Búsqueda local

Con el objetivo de dotar de la capacidad de .ªprendizaje.ª los individuos, se ha diseñado un algoritmo de optimización basado en búsquedas locales. El algoritmo implementado posee un vector inicial de booleanos inicializados a *false*, haciendo referencia a la técnica de *Don't look bits* [DLB]. Por lo tanto se trata de un algoritmo de búsqueda local consistente en *Primero el mejor*.

El algoritmo propuesto realizará transposiciones *2-opt* para generar nuevas soluciones con el fin de obtener soluciones con coste menor a la actual, partiendo inicialmente de cada una de las soluciones correspondientes a los individuos de la población del algoritmo genético.

En este algoritmo se establece un criterio de parada consistente en un número máximo de iteraciones (400), o cuando se ha explorado todo el vecindario y no se ha encontrado una solución mejor a la actual. En caso de no encontrar soluciones prometedores, se podría afirmar que la solución se encuentra en un mínimo local.

El objetivo de implementar este algoritmo es el de poder realizar búsquedas locales a los individuos de la población con el fin de que aprendan sobre su entorno, y dependiendo del enfoque propuesto conservar dicha información o incluso poder transmitirla a las posteriores generaciones.

3.4. Variante Baldwiniana

El enfoque Baldwiniano consiste en la dotación de aprendizaje por parte de los individuos sobre su entorno, de forma que estos puedan alcanzar posibles mínimos locales y guiarse por esta información de cara a la exploración del algoritmo. Sin embargo, los individuos poseen los mismos genes, y no se transmite este nuevo aprendizaje a los hijos, por lo que la información aprendida sirve de guía, pero no como información genética.

Por lo tanto cabe destacar de que la única información que variará en este enfoque de los diferentes individuos es la capacidad de aprendizaje y guía de la población en base a este aprendizaje, pero se trata de un aprendizaje temporal, por lo que aunque puede servir de guía, también es posible que dicho aprendizaje no se considere por la población y se pierda con el paso de las generaciones.

3.5. Variante Lamarckiana

A diferencia del anterior enfoque, en este caso el aprendizaje obtenido se transmite en las diferentes generaciones. Este enfoque consiste en la capacidad de transmisión de aprendizaje entre generaciones, ya que los individuos a lo largo de su existencia son capaces de aprender, y dicha información es transmitida genéticamente a sus hijos.

Con este enfoque cabe destacar que es más probable caer en mínimos locales, a la vez que permite explorar soluciones más prometedoras, y que las siguientes generaciones o bien se estanquen o bien sigan explorando el camino indicado por sus padres a soluciones prometedoras.

Por lo tanto, cada generación es optimizada previa a su comportamiento general, y esta optimización además de ser un procedimiento costoso (que se traduce en una mayor cantidad de tiempo de cómputo y coste).

4. Ejecución de la implementación

La implementación ha sido realizada en el lenguaje *C++*, utilizando como gestor de tareas la herramienta *Makefile*, por lo que existen las diferentes tareas, que se ejecutan de la siguiente forma:

- **make**: Construye el proyecto y genera el ejecutable de la práctica.
- **make clean**: Limpia los ficheros *.o*, es decir, los objetos.
- **make mrproper**: Además de la funcionalidad de *make clean*, limpia el ejecutable generado.

Tras ejecutar la orden **make**, y generar el ejecutable, para ejecutar se debe seguir el formato:

./Practica_QAP <num_algoritmo><fichero_datos><num_semilla>

Los algoritmos considerados son:

1. AGG con cruce posición.
2. AGG Baldwiniano con cruce posición.
3. AGG Lamarckiano con cruce posición.
4. AGG con cruce OX.
5. AGG Baldwiniano con cruce OX.
6. AGG Lamarckiano con cruce OX.
7. AGE con cruce posición.
8. AGE Baldwiniano con cruce posición.
9. AGE Lamarckiano con cruce posición.
10. AGE con cruce OX.
11. AGE Baldwiniano con cruce OX.
12. AGE Lamarckiano con cruce OX.

Por lo tanto, si se desea ejecutar el programa con un algoritmo *AGG con cruce posición*, para el fichero *tai256c.dat* con semilla 1, la ejecución sería:

./Practica_QAP 1 tai256c.dat 1

5. Resultados

Los resultados que se estudiaran a continuación han sido obtenidos mediante diferentes ejecuciones del algoritmo con distintas semillas, con el fin de reducir en la medida de lo posible la aleatoriedad de los algoritmos. El número de semillas probadas por cada algoritmo es de 30 semillas, lo que equivale a 30 ejecuciones y 30 resultados diferentes por cada algoritmo.

Los resultados se promediarán y se obtendrán además los promedios de tiempos de ejecución. Posteriormente se analizarán los mejores resultados obtenidos.

5.1. Pruebas realizadas

Las pruebas realizadas en los diferentes algoritmos poseen los siguientes parámetros:

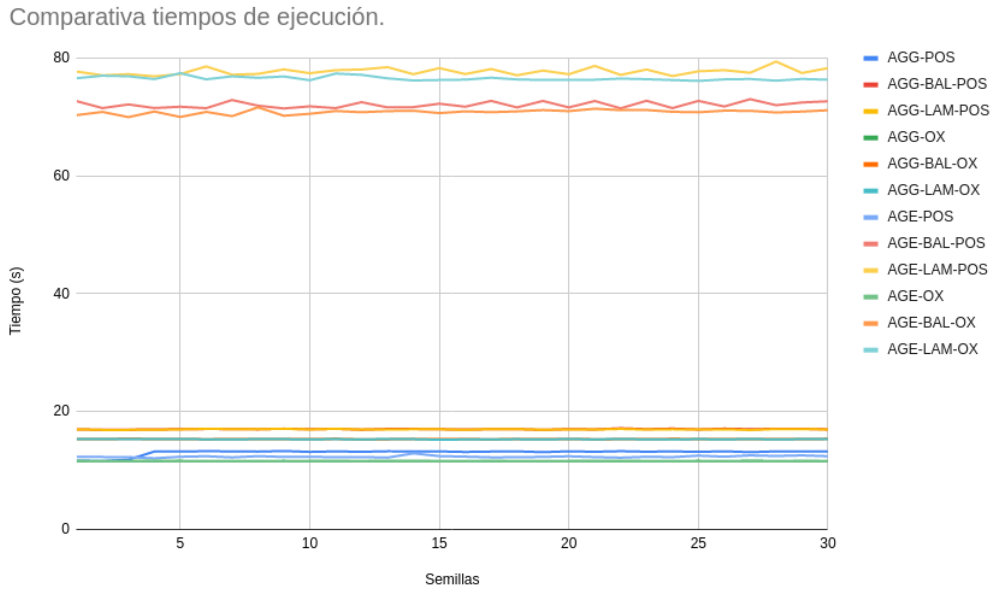
- **Estrategia constructiva**: Generación aleatoria de primera población.
- **Estrategia de transposición**: 2-opt.
- **Tamaño de la población**: 50 individuos.
- **Operador de selección**: Torneo binario.
- **Operador de cruce**:
  - *Operador por posición.*
  - *Operador OX*: Tamaño de subcadena de 42 (256/6) elementos.
- **Mutación**: Probabilidad de 0.001 %.
- **Reemplazo**: Elitismo, sustitución del peor miembro.
- **Condición de parada**: 50.000 iteraciones (llamadas a la función objetivo).

5.2. Datos obtenidos

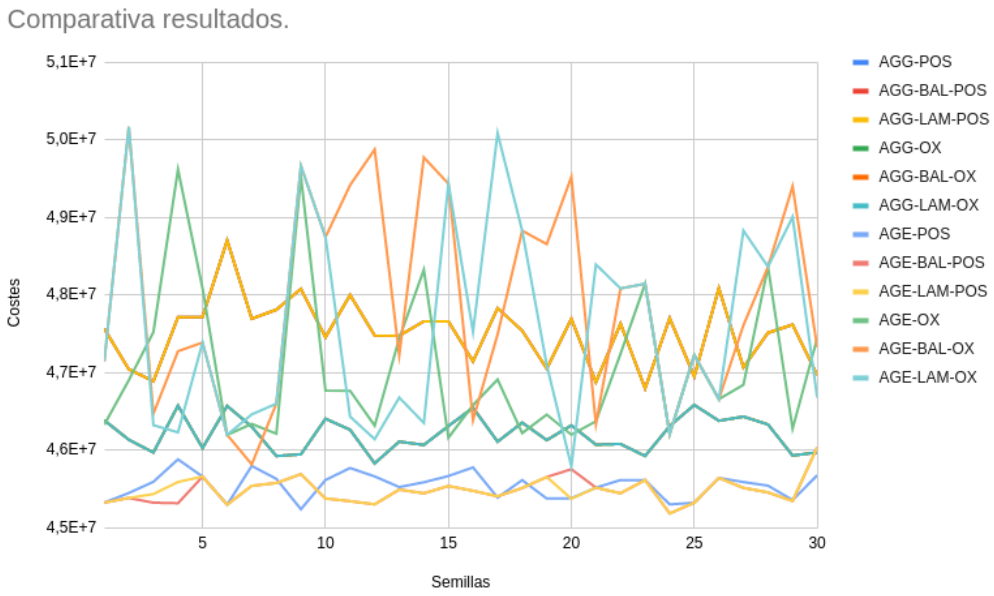
A continuación se muestran los mejores resultados obtenidos por cada algoritmo:

Algoritmo	**Coste**
AGG con cruce por posición.	46794702
AGG Baldwiniano con cruce por posición.	46794702
AGG Lamarckiano con cruce por posición.	46794702
AGG con cruce OX.	45829092
AGG Baldwiniano con cruce OX.	45829092
AGG Lamarckiano con cruce OX.	45829092
AGE con cruce por posición.	45237544
AGE Baldwiniano con cruce por posición.	45183274
AGE Lamarckiano con cruce por posición.	45183274
AGE con cruce OX.	46160288
AGE Baldwiniano con cruce OX.	45815956
AGE Lamarckiano con cruce OX.	45790394

Por otro lado, se puede observar con el promedio de tiempos tras 30 ejecuciones de cada algoritmo los diferentes tiempos de ejecución:



Por último, se realiza una comparativa de los costes obtenidos por cada algoritmo:



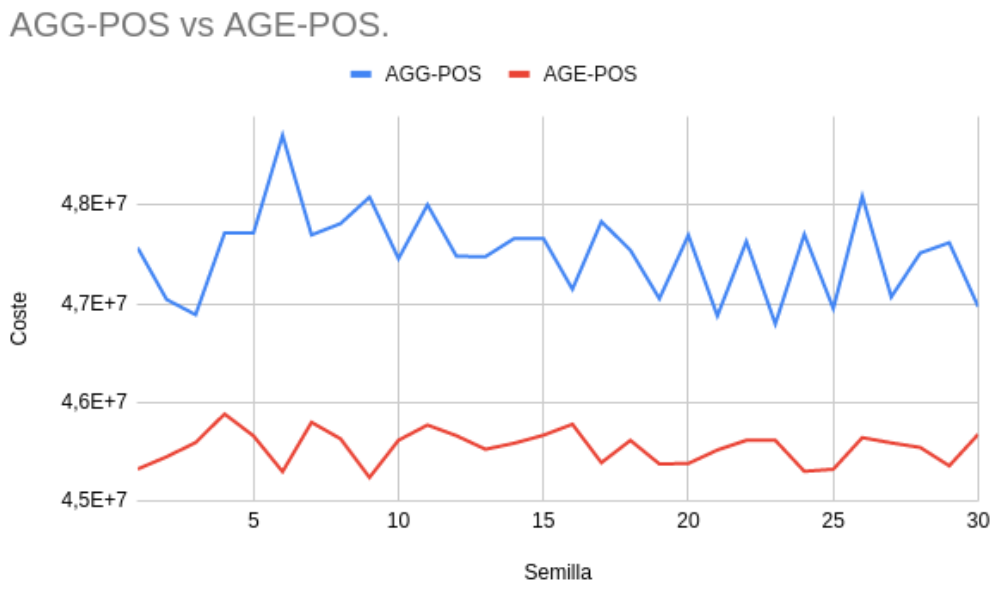
Se obtienen finalmente como los mejores resultados los siguientes con su respectivo algoritmo y tiempo de ejecución asociado:

Coste	Algoritmo	Tiempo
45183274	AGE-BAL-POS	71,5433 segundos
45183274	AGE-LAM-POS	76,9891 segundos
45237544	AGE-POS	12,2673 segundos
45296254	AGE-POS	12,3852 segundos
45296254	AGE-BAL-POS	71,5295 segundos
45296254	AGE-LAM-POS	78,6229 segundos
45299946	AGE-POS	12,2328 segundos

5.3. Análisis

Dentro de los diferentes aspectos a analizar sobre los algoritmos evolutivos implementados, cabe destacar que existe una tendencia a obtener mejores resultados en los enfoques estacionarios, esto se debe a que sus características permiten de por sí una mayor explotación y exploración de los posibles valores, a diferencia del enfoque generacional, el cual se centra más en la exploración, por lo que realmente cabe esperar que los resultados obtenidos en algoritmos con enfoque estacionario sean por norma general mejores.

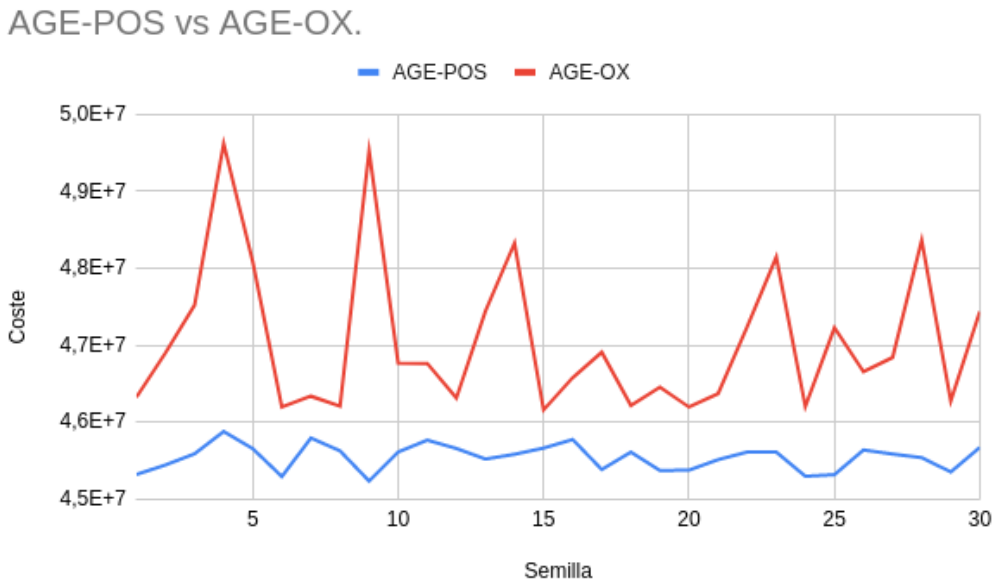
A continuación se muestra una gráfica con los resultados obtenidos por los algoritmos más basicos utilizando el operador de cruce de posición:



Se puede observar que por norma general (no se puede afirmar que siempre debido a la pequeña cantidad de pruebas y ajustes de parámetros posibles) para este problema se obtienen mejores resultados con un enfoque estacionario que realiza una mayor explotación de los posibles resultados, mientras que el enfoque generacional alcanza más mínimos locales.

Por otro lado, la importancia del operador de cruce es vital en este problema, ya que añaden capacidad de exploración y explotación, de forma que el cruce por posición ofrece una mayor exploración debido a la aleatoriedad del mismo, mientras que el cruce OX ofrece una mayor explotación al utilizar los genes de ambos padres. Se considera que por su capacidad exploratoria, para este caso el operador de cruce es el más adecuado para el problema.

A continuación se muestra una gráfica con los resultados obtenidos por los enfoques estacionarios haciendo uso de ambos operadores de cruce:

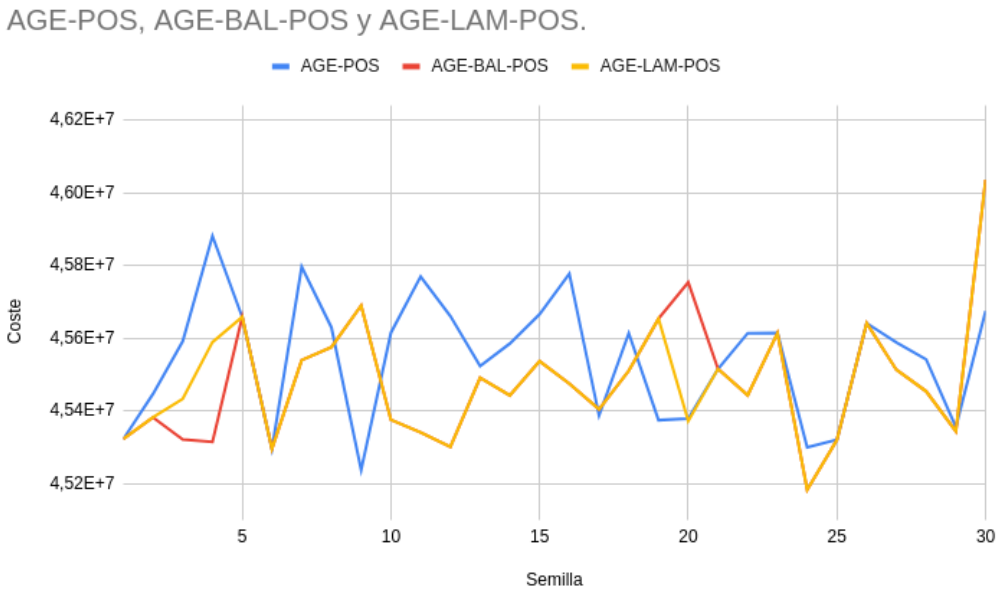


Se puede observar que por norma general se obtienen mejores resultados con el operador de cruce por posición, debido a la capacidad exploratoria que esta añade al enfoque estacionario, combinando la explotación del enfoque con la exploración del operador.

En lo que hace referencia a las variantes Baldwiniana y Lamarckiana, cabe destacar que el aprendizaje, como en cualquier problema puede conllevar beneficios y riesgos a la vez, ya que la exploración de zonas más prometedores puede llevar a obtener mejores soluciones, añadiendo una mayor cantidad de cómputo y tiempo de ejecución, pero a su vez, puede reducir la capacidad exploratoria del algoritmo, por lo que en ciertos casos puede llegar a producir peores resultados por caer en mínimos locales y no conseguir escapar de estos.

Dentro de los resultados obtenidos además se ha podido comprobar que en la mayoría de los casos no existe una gran diferencia entre los resultados obtenidos entre ambas variantes, lo cual se debe a que la transmisión del aprendizaje en generaciones puede suponer una ventaja en ciertos aspectos, pero en la mayoría de los casos con que una población posea un aprendizaje es suficiente para que esta se oriente hacia zonas prometedoras sin necesidad de que se trasladen sus individuos y sus hijos se vean afectados por este aprendizaje heredado.

A continuación se muestra una gráfica que compara los resultados obtenidos por el algoritmo genético estacionario con cruce por posición frente a sus variantes:



Como es de esperar, los resultados obtenidos por la variante Lamarckiana obtiene por norma general mejores resultados, aunque no siempre, ya que en ciertos casos puede caer en mínimos locales, pero sí tiene una tendencia a obtener mejores resultados. Por otro lado se puede observar lo previamente comentado, la variante Baldwiniana puede llegar en su mayoría a resultados similares aunque en ciertos casos podrá comportarse de forma diferente al no heredar la información entre poblaciones.

6. Conclusiones

Durante la realización de la práctica se han podido evaluar como se comportan los diferentes algoritmos y se ha permitido llegar a una serie de conclusiones:

- Para el problema dado, las mejores soluciones se obtienen utilizando por norma general un operador de cruce por posición debido a la **aleatoriedad** que ofrece, el cual da lugar a una mayor **exploración** del entorno.
- Entre los enfoques vistos, se obtienen mejores resultados con el enfoque **estacionario** ya que a diferencia del **generacional**, este primero conserva más tiempo las soluciones y permite explotarlas en mayor medida, mientras que el segundo enfoque se centra más en la exploración y no llega a explotar de la misma forma las soluciones obtenidas.
- Las variantes, pese a aportar numerosas ventajas, conllevan en el enfoque estacionario una mayor cantidad de cómputo que se traduce en un tiempo de ejecución casi 6 veces superior al presentado por la versión básica del algoritmo.
- Las variantes **Baldwiniana** y **Lamarckiana** ofrecen mejores resultados, y de forma similar, concluyendo con que en promedio mejoran el algoritmo básico, y dentro del problema, existen casos en los que se adaptan mejor una versión que la otra. Sin embargo, cabe destacar que la versión **Baldwiniana** es ligeramente más rápida, mientras que la versión **Lamarckiana** permite explotar en mayor medida conservando las soluciones exploradas.

Por último, cabe concluir indicando que al existir un determinado factor de aleatoriedad, debido a que se tratan de algoritmos estocásticos, el factor aleatorio influye en los resultados obtenidos, por lo que podrían llegar a obtenerse mejores soluciones con otras posibles semillas, pero en este problema únicamente se han valorado 30 ejecuciones independientes, por lo que realmente hacer una afirmación general resultaría más complejo, pero si se puede obtener una buena aproximación con los resultados valorados.

7. Bibliografía

- [BIO] Apuntes asignatura Bioinformática UGR curso 2013-2014.
- [MEH] Apuntes asignatura Metaheurísticas UGR curso 2020-2021.
- [IC] Apuntes asignatura Inteligencia Computacional UGR curso 2020-2021.
- [DLB] TSP Basics: Don't Look Bits (DLB) – general idea.
- [BER] Fernando Berzal, *Algoritmos Genéticos*. Departamento *Decsai*.