UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Inteligencia computacional

Práctica de algoritmos evolutivos

Resolución de problemas NP

QAP

Curso 2016-2017

Máster en Ingeniería Informática

Ayhami Estévez Olivas

aythae@correo.ugr.es

Índice de contenido

Índice de figuras

Índice de tablas

Introducción

Los algoritmos genéticos se basan en la aplicación de los principios de la evolución biológica a problemas de optimización o búsqueda en los que el espacio de soluciones es grande y no se conoce un algoritmo concreto para resolver el problema. Por ello representan las posibles soluciones al problema como individuos de una población los cuales son seleccionados para reproducirse, se reproducen creando nuevos individuos y sufren mutaciones. En general, el objetivo de cualquier algoritmo genético es encontrar una solución óptima para una cierta función objetivo [DELGADO CALVO-FLORES, 2014], aunque habitualmente basta con encontrar una solución lo suficientemente adecuada ya que algunos problemas pueden no tener una solución óptima conocida.

Por ello se emplean frecuentemente en problemas NP como es el caso de la asignación cuadrática o QAP (Quadratic Assignment Problem), este problema se puede expresar como: “Supongamos que queremos decidir dónde construir *n* instalaciones (p.ej. fábricas) y tenemos *n* posibles localizaciones en las que podemos construir dichas instalaciones. Conocemos las distancias que hay entre cada par de instalaciones y también el flujo de materiales que ha de existir entre las distintas instalaciones (p.ej. la cantidad de suministros que deben transportarse de una fábrica a otra). El problema consiste en decidir dónde construir cada instalación de forma que se minimice el coste de transporte de materiales. Formalmente, si llamamos *d(i, j)* a la distancia de la localización *i* a la localización *j* y *w(i, j)* al peso asociado al flujo de materiales que ha de transportarse de la instalación *i* a la instalación *j*, hemos de encontrar la asignación de instalaciones a localizaciones que minimice la función de coste

$$\sum_{i,j} w(i,j)d(p(i),p(j))$$

Ilustración 1: Función a optimizar en el problema QAP [BERZAL GALIANO, 2016]

donde *p()* define una permutación (solución o individuo) sobre el conjunto de instalaciones”.[BERZAL GALIANO, 2016]

Implementación

Lenguaje de programación seleccionado

Me decidí a realizar una implementación en Java tras buscar cuales son mejores lenguajes para la implementación de algoritmos genéticos. En esa búsqueda di con el articulo [MERELO-GUERVÓS, 2016] que determina tras probar números lenguajes de programación y estructuras de datos que Java es el que ofrece un rendimiento mejor, además de esto Java es el lenguaje en el que me siento más cómodo programando debido a la experiencia que tengo con él.

Representación del problema

Teniendo en cuenta las características del problema y los comentarios hechos en clase para representar cada una de las soluciones al problema elegí un vector de enteros del tamaño del problema concreto, los elementos de este vector van numerados entre 0 y el tamaño del problema de acuerdo a como se manejan los arrays en Java y en la mayoría de los lenguajes de programación, es necesario que no existan números duplicados en una permutación para considerarla valida. Esta permutación representa el orden de multiplicación de la matriz de distancias tal y como se muestra en la Ilustración 1.

Materiales de apoyo empleados

Además de los apuntes de practicas correspondientes a esta asignatura, en concreto el tema de Algoritmos Genéticos de Fernando Berzal [BERZAL GALIANO, 2014], he utilizado un tutorial de algoritmos genéticos para novatos en Java que explica e implementa un algoritmo genético básico, lo he utilizado para empezar a plantear mi esquema de clases antes de tener muy claro como funcionaba un algoritmo genético [JACOBSON, 2012]. Por último he usado el libro "*Genetic Algorithms in Java Basic"*[JACOBSON & KANBER, 2015] que resuelve el problema del viajante comercio explicando cuales son los mejores operadores para este, según se dice en [BERZAL GALIANO, 2016] el problema del viajante de comercio puede interpretarse como un caso particular del QAP, por ello pensé que las soluciones validas para este también lo serían para el QAP. Esto me motivo a seleccionar los mecanismos de selección, mutación y cruce empleados en mis implementaciones.

Algoritmo genético estándar

Comencé la implementación del algoritmo genético básico a partir del siguiente pseudocódigo que se aporta en [BERZAL GALIANO, 2014]:

```
t ← 0
población(t) ← poblaciónInicial
EVALUAR(población(t))
while not (condición de terminación)
     t ← t + 1
     población(t) ← SELECCIONAR(población(t-1))
     población(t) ← CRUZAR(población(t))
     población(t) ← MUTAR(población(t))
     EVALUAR(población(t))
return población(t)
```

La función de evaluación a optimizar que devuelve el fitness de los individuos es la que se representa en la Ilustración 1, por lo que faltan por determinar un mecanismo de selección, un operador de cruce, un operador de mutación, un mecanismo de reemplazo generacional y una condición de terminación.

Mecanismo de remplazo (selección de supervivientes)

Según se dice en [BERZAL GALIANO, 2014] se ha demostrado que para que un algoritmo genético converja en un óptimo global es necesario que tenga cierto componente de elitismo, es decir que el mejor individuo de una población pase a la siguiente directamente.

Como modelo concreto he elegido un modelo generacional, en el que cada individuo vive durante una única generación, con elitismo donde los mejores individuos se pasan directamente a la siguiente generación. En general selecciono a los dos mejores individuos distintos de cada generación para salvarse, intentando con ello obtener individuos en diversas zonas del espacio de soluciones cercanos a óptimos locales, ya que en caso de seleccionar un solo individuo es fácil que este caiga en un óptimo local del que solo se escaparía mediante una mutación afortunada.

Mecanismo de selección

Como operador de selección busque en diversos papers que intentan resolver el problema QAP y en general se emplean 2 mecanismos por encima de los demás: mecanismo de selección proporcional ruleta y selección por torneo. El primero le asigna una probabilidad de cruce a un individuo en función de su fitness y usa esas probabilidad para seleccionar aleatoriamente individuos, mientras que el segundo selecciona de manera totalmente aleatoria k individuos y de esos selecciona el mejor.

Teniendo en cuenta que los algoritmos genéticos son algoritmos no deterministas y que a la hora de la verdad la diferencia entre un buen y un mal resultado muchas veces es el azar decidí probar con la selección por torneo. Para aplicar este método hay que determinar el tamaño del torneo (k), para probar decidí seleccionar 5.

Operador de cruce

Siguiendo los consejos de [JACOBSON & KANBER, 2015] es necesario seleccionar un operador de cruce que conserve el orden de los genes ya que en este problema el orden de los genes determina la solución. Ellos utilizan un operador llamado cruce ordenado que buscando en papers sobre la solución del QAP ese operador no dejaba de aparecer así que decidí implementarlo. Este operador se basa en seleccionar aleatoriamente dos puntos de los padres y traspasar directamente el “subcromosoma”que se encuentra entre dichos puntos de uno de los padres al cromosoma hijo, véase Ilustración 2.

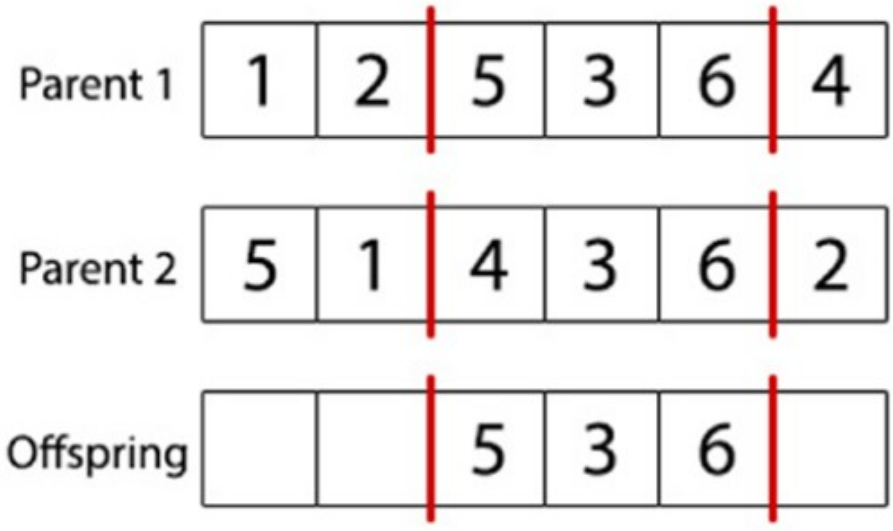

Ilustración 2: Cruce ordenado 1 [JACOBSON & KANBER, 2015]

Una vez hecho esto para rellenar el resto del cromosoma hijo se utiliza al otro padre recorriéndolo desde la segunda posición de corte aleatoria en adelante e insertando en orden (desde el primer gen libre) los alelos que no estén ya en el cromosoma hijo, en este ejemplo concreto se insertaría primero el 2 (último elemento del padre 2), luego se intentaría insertar el 5 (primer elemento del padre 2) pero al ya encontrarse este en el hijo por haber sido heredado del padre 1 se saltaría al siguiente y se insertaría el 1. Sucesivamente se iría insertando así hasta llegar al resultado que se observa en la Ilustración 3.

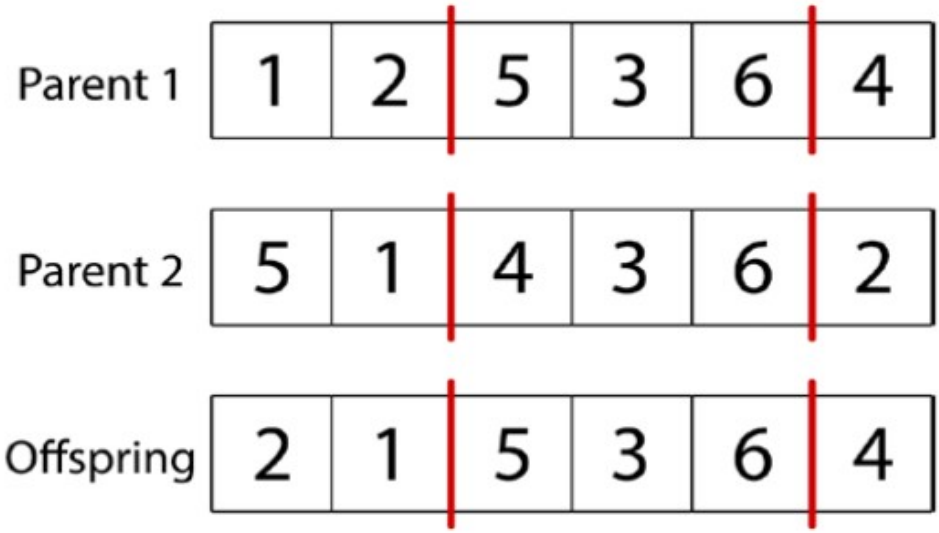

Ilustración 3: Cruce ordenado 2 [JACOBSON & KANBER, 2015]

Para sacar algo más de rendimiento a este proceso genero otro hijo de los mismos padres pero invirtiendo el padre 1 y el 2.

Operador de mutación

Siguiendo los consejos dados en clase y los que se dan en [JACOBSON & KANBER, 2015] es necesario usar un operador de mutación que no cree permutaciones no validas y teniendo en cuenta que no puede haber números repetidos parece sencillo pensar que un operador de mutación que intercambie un número por otro sería lo más indicado. Por ello se eligen aleatoriamente dos genes que intercambiaran sus alelos como muestra en la Ilustración 4. La probabilidad de seleccionar un gen para mutarlo se rige por una tasa o probabilidad de mutación de gen fijada a nivel global para cada ejecución del algoritmo. Los mejores resultados experimentales se han obtenido con tasas entre 0,2 % y 0,3%

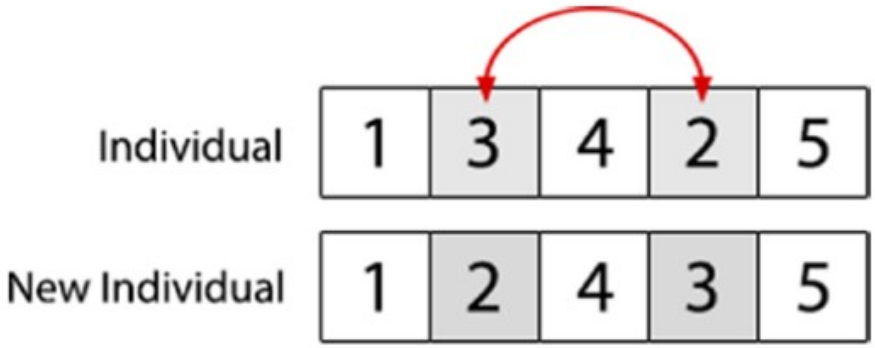

Ilustración 4: Operador de mutación por intercambio [JACOBSON & KANBER, 2015]

Condición de terminación

Como condición de terminación del algoritmo he implementado dos posibilidades:

- Ejecutar el algoritmo un número fijo de veces
- Ejecutar el algoritmo hasta que se alcance un número fijo de generaciones sin mejorar (podemos deducir que se ha atascado en un máximo local)

Dichas opciones se dejan a la elección del usuario en cada ejecución del algoritmo, pero como es obvio se obtienen mejores resultados (aunque tarde más) con la segunda.

Resultados

A continuación se presentan los mejores resultados obtenidos con el algoritmo estándar sobre el problema objetivo *tai256c.* En la Tabla 1 se representan sus parámetros de ejecución y en la Ilustración 5 se observa una gráfica de la evolución del fitness mínimo respecto al paso de las generaciones

Tamaño de la población	100
Tamaño del torneo	5
Probabilidad de mutación de los genes	0,002
Individuos elitistas	2
Generación de finalización	958
Coste de la mejor solución	**45.685.504**
Diferencia respecto a la solución óptima	2,069%
Tiempo de ejecución	9.366 ms
Tiempo medio por generación	9,777 ms

Tabla 1: Parámetros de ejecución del algoritmo estándar

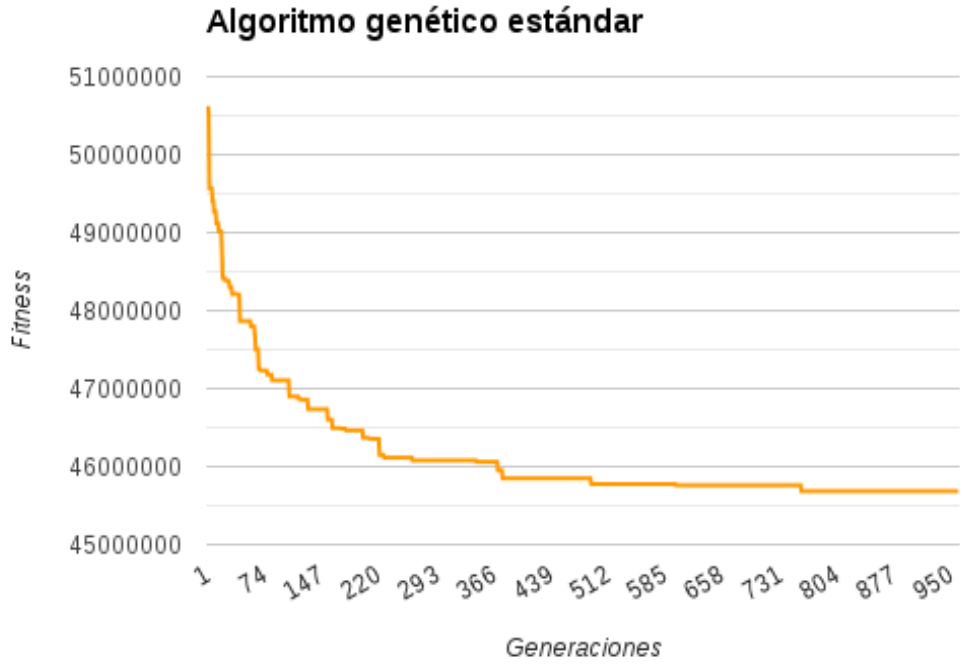

Ilustración 5: Evolución del fitness respecto de las generaciones en el algoritmo estándar

Variante baldwiniana

A partir de los operadores y mecanismos definidos en el apartado anterior se incorporan técnicas de optimización local para hacer que los individuos de una población puedan “aprender”. Esto se basa en aplicar un algoritmo de optimización local en cada individuo para obtener el mayor fitness que puede alcanzar este mediante la mutación de sus genes. Dicho fitness optimizado se le asigna al individuo,

pero su material genético se mantiene intacto (sin aplicar las mejoras de la optimización local) [BERZAL GALIANO, 2016].

Respecto al algoritmo de optimización local he utilizado como base el algoritmo 2-opt cuyo pseudocódigo aparece en [BERZAL GALIANO, 2016] combinado con un cálculo del fitness optimizado a partir del fitness previo y las posiciones de la mutación para ahorrar operaciones. Además para mejorar el rendimiento aún más he paralelizado esta optimización que se lleva a cabo al calcular el fitness de la población para que se realice en múltiples hilos (2 o 4) dependiendo de las capacidades del ordenador en el que se ejecute el algoritmo.

Empíricamente he comprobado que resulta más eficiente aplicar la optimización baldwiniana a parte de la población en lugar de a toda la población, tanto en términos de tiempo de ejecución (como es obvio) como en términos de resultados ya que al convivir individuos estándar con balwinianos se incrementa la variedad de individuos que siempre es buena para escapar de óptimos locales.

Resultados

A continuación se presentan los mejores resultados obtenidos con el algoritmo baldwiniano sobre el problema objetivo *tai256c.* En la Tabla 2 se representan sus parámetros de ejecución y en la Ilustración 6 se observa una gráfica de la evolución del fitness mínimo respecto al paso de las generaciones.

Tamaño de la población	60
Tamaño del torneo	6
Probabilidad de mutación de los genes	0,0025
Probabilidad de optimización balwiniana	0,5
Individuos elitistas	2
Generación de finalización	127
Coste de la mejor solución	**44.835.526**
Diferencia respecto a la solución óptima	0,17%
Tiempo de ejecución	994.890 ms
Tiempo medio por generación	7.833,779 ms

Tabla 2: Parámetros de ejecución del algoritmo baldwiniano

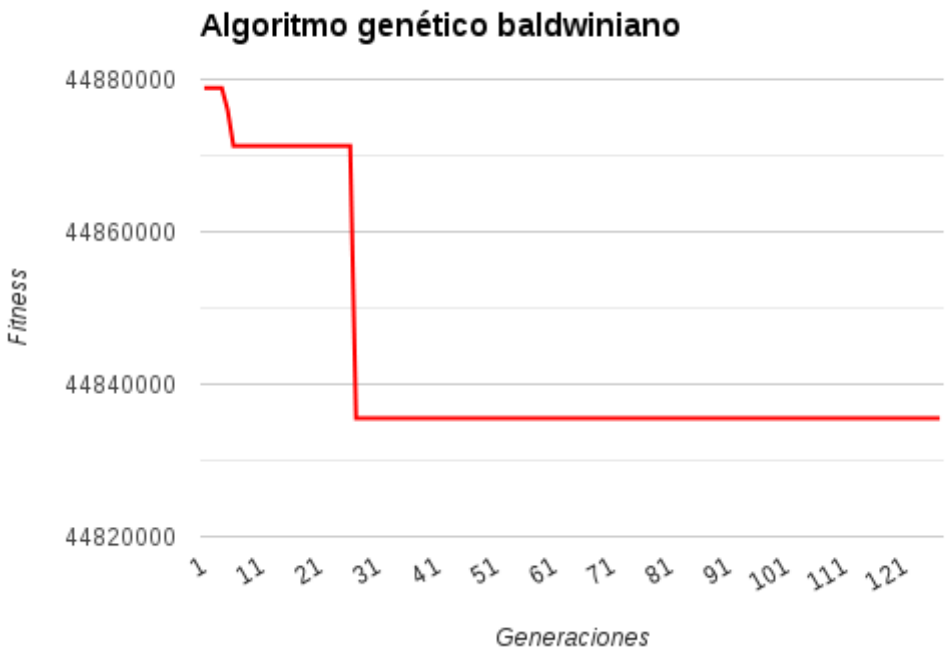

Ilustración 6: Evolución del fitness respecto de las generaciones en el algoritmo baldwiniano

Variante lamarckiana

A partir de los operadores y mecanismos definidos en el algoritmo estándar se incorporan técnicas de optimización local para hacer que los individuos de una población puedan "aprender" y permitir que se herede lo "aprendido. Esto se basa en aplicar un algoritmo de optimización local en cada individuo para obtener el fitness igual que en la variante balwiniana pero además el material genético del individuo se sustituye por la versión optimizada, de modo que se transmite a los descendientes [BERZAL GALIANO, 2016].

El algoritmo de optimización es el mismo que en el caso de la variante baldwiniana, pero como ya se ha dicho ahora se optimiza tanto el fitness como el cromosoma de los individuos.

Empíricamente he comprobado que resulta más eficiente aplicar la optimización lamarckiana a parte de la población en lugar de a toda la población como ocurre con la optimización baldwiniana.

Resultados

A continuación se presentan los mejores resultados obtenidos con el algoritmo lamackiano sobre el problema objetivo *tai256c.* En la Tabla 3 se representan sus parámetros de ejecución y en la Ilustración 7 se observa una gráfica de la evolución del fitness mínimo respecto al paso de las generaciones

Tamaño de la población	60
Tamaño del torneo	6
Probabilidad de mutación de los genes	0,003
Probabilidad de optimización lamackiana	0,5
Individuos elitistas	2
Generación de finalización	378
Coste de la mejor solución	**44.810.528**
Diferencia respecto a la solución óptima	0,114%
Tiempo de ejecución	2.299.089 ms
Tiempo medio por generación	6.082,246 ms

Tabla 3: Parámetros de ejecución del algoritmo lamackiano

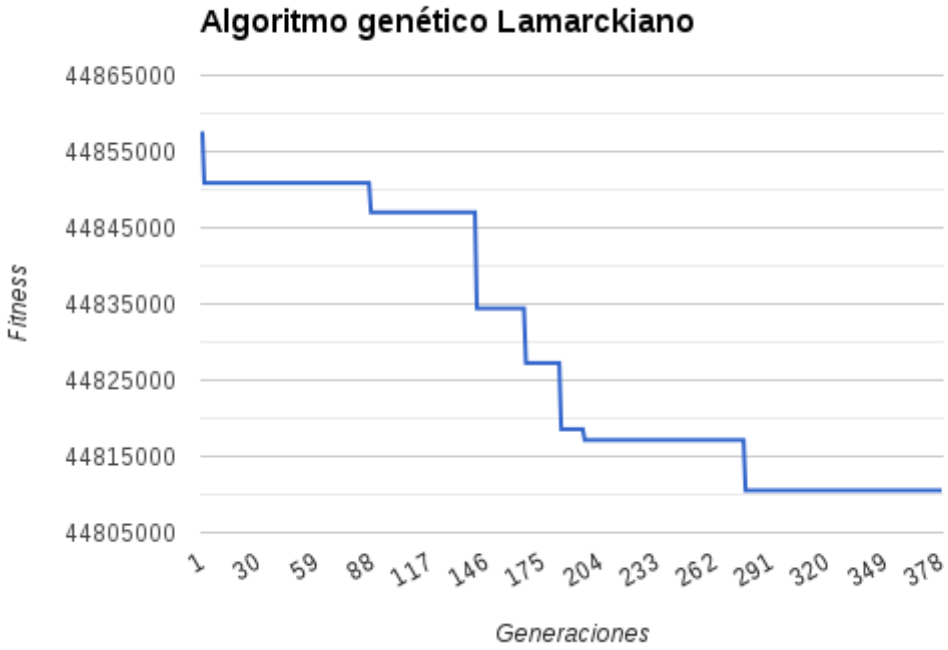

Ilustración 7: Evolución del fitness respecto de las generaciones en el algoritmo lamackiano

Análisis de resultados

Comparación de variantes optimizadas del algoritmo

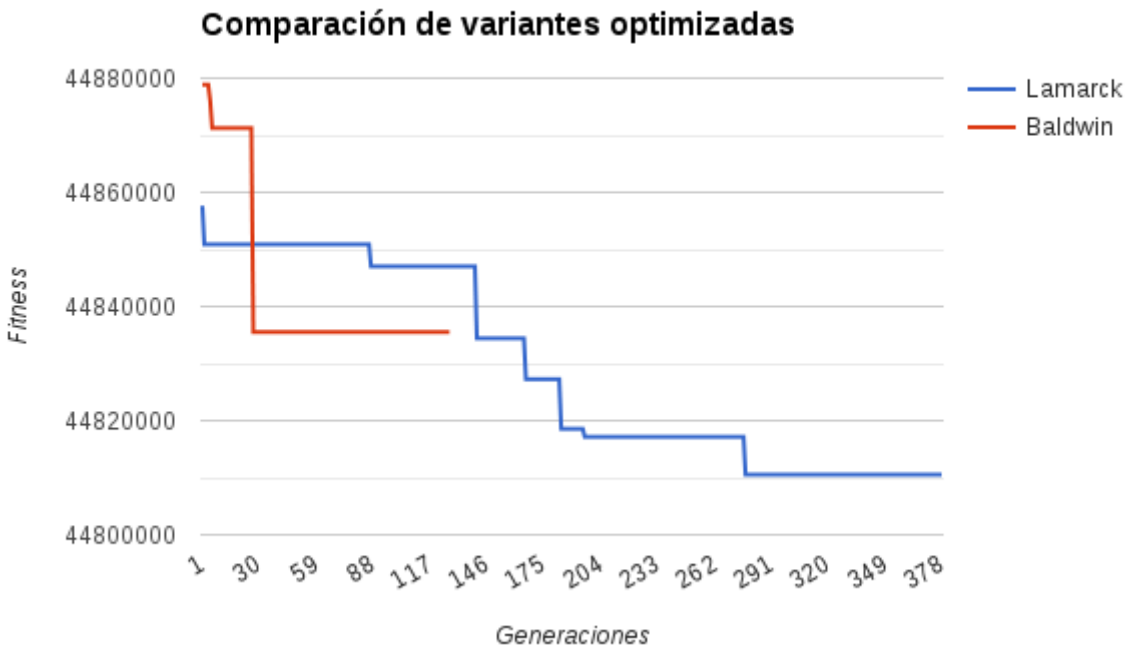

Ilustración 8: Evolución del fitness respecto de las generaciones en los algoritmos optimizados

Como se puede ver la variante baldwiniana parece converger más rápido, pero se atasca en un máximo local con más facilidad que la lamackiana que obtiene los mejores resultados.

Comparación entre todas las variantes

A modo de conclusión aporto un gráfico en el que se pueden ver enfrentadas todas variantes del algoritmo genético. Debido a la escala tan diferente de los algoritmos optimizados y el estándar no se aprecian bien estos últimos, por ello se aportan las gráficas previas en las que se pueden ver estas variantes optimizadas en detalle.

Como se puede apreciar en la siguiente ilustración las variables optimizadas del algoritmo son superiores en todos los casos desde la primera generación a la mejor ejecución del algoritmo estándar.

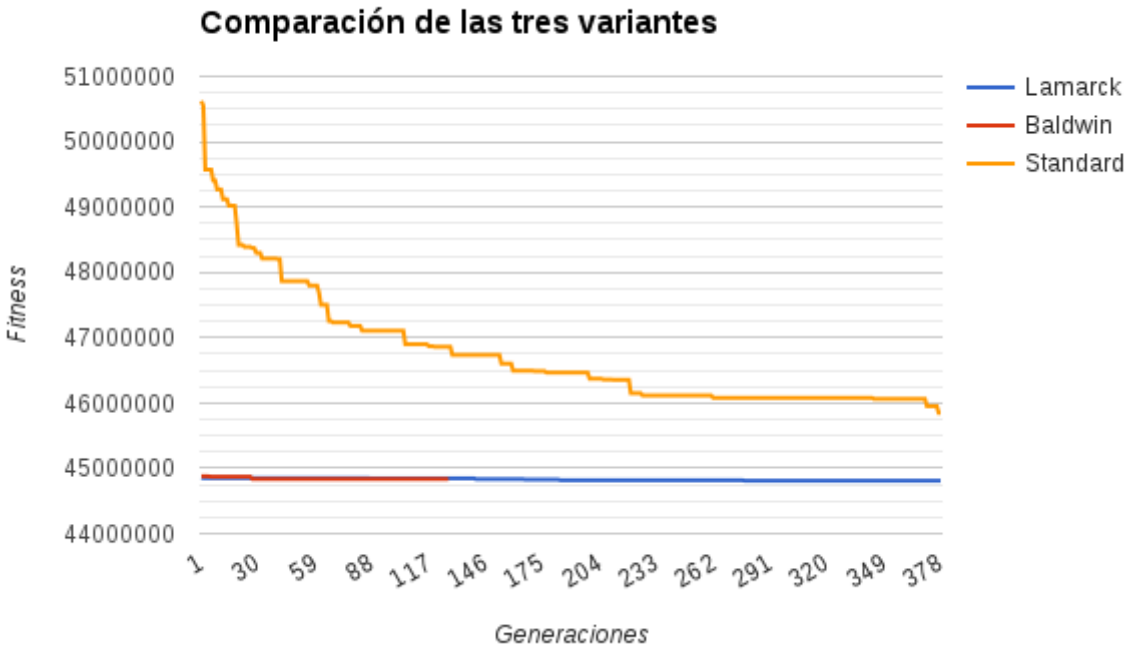

Ilustración 9: : Evolución del fitness respecto de las generaciones en las tres variantes del algoritmo

Diferencia de tiempos entre variantes

Es necesario mencionar que además de los resultados de las variantes optimizadas y la estándar hay una gran diferencia de tiempos de ejecución entre estas causadas por el algoritmo de optimización local 2-opt ya que tiene una eficiencia $O(n^2)$ donde n es el tamaño del problema a evaluar. Esto hace que se pase de una media de 9,777 ms por generación en el caso estándar a 6.082,246 ms por generación como poco en el caso de las variantes optimizadas.

Conclusiones

Como se puede ver a lo largo de esta memoria los algoritmo genéticos son algoritmos no deterministas con un componente aleatorio muy importante, esto hace que no haya dos ejecuciones idénticas aunque no se cambie ningún parámetro. Esto que puede parecer una desventaja muy importante para cualquier algoritmo permite obtener soluciones “aceptables” a problemas demasiado complejos como para que exista un algoritmo determinista que finalice en un tiempo aceptable. Por ello son siempre una posible solución ante problemas complejos.

Respecto a las versiones optimizadas de estos algoritmos no se contemplado mucha diferencia, la versión lamarckiana parece arrojar mejores resultados pero teniendo en cuenta el componente aleatorio es posible que se deba al azar y a como se ha realizado la implementación concreta.

Al igual que ocurría para ajustar los hiperparámetros de las redes neuronales, los cuales se elegían siguiendo algunas recomendaciones heurísticas sin que hubiera ninguna prueba teórica de que algunos son mejores que otros para ciertos problemas. Lo mismo pasa con los parámetros de un algoritmo genético, solo se pueden realizar pruebas empíricas para afinarlos y confiar en la experiencia de los desarrolladores.

Bibliografía

DELGADO CALVO-FLORES, 2014: Miguel Delgado Calvo-Flores, Apuntes de Inteligencia Computacional, 2014

BERZAL GALIANO, 2016: Fernando Berzal Galiano, Práctica de algoritmos evolutivos. QAP. Enunciado, 2016

MERELO-GUERVÓS, 2016: Juan-Julian Merelo-Guervós et al., Ranking the Performance of Compiled and Interpreted Languages in Genetic Algorithms, 2016

BERZAL GALIANO, 2014: Fernando Berzal Galiano, Algoritmos Genéticos, 2014

JACOBSON, 2012: Lee Jacobson, Creating a genetic algorithm for beginners, consultado 01/2017, http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3

JACOBSON & KANBER, 2015: Lee Jacobson & Burak Kanber, Genetic Algorithms in Java Basics, 2015