UNIVERSIDAD DE GRANADA

Master en Ingeniería Informática

Inteligencia Computacional

Práctica de algoritmos evolutivos

Resolución de problemas NP

QAP

Marvin Matías Agüero Torales

maguero@correo.ugr.es

2016-17

Índice

Introducción

El objetivo de la práctica es resolver un problema de optimización típico utilizando técnicas de computación evolutiva. Se implementaran algunas variantes de algoritmos evolutivos para resolver el problema de la asignación cuadrática o QAP, un problema fundamental de optimización combinatoria con numerosas aplicaciones.

Por su interés teórico y práctico, existe una variedad muy amplia de algoritmos que abordan la resolución del QAP, pero con algoritmos evolutivos se evaluará su rendimiento sobre instancias concretas del problema.

Implementación

Los casos de prueba para comprobar el funcionamiento de las distintas heurísticas *greedy* se han obtenido de la biblioteca QAPLIB (Burkard, Çela, Karisch, & Rendl, 1997). El formato de los archivos de datos es el siguiente:

n, tamaño del problema

A, matriz de flujos

B, matriz de distancia

Se trabajará sobre el conjunto tai256c de QAPLIB (Burkard et al., 1997), la mejor solución obtenida con un algoritmo evolutivo para este problema tiene un coste de 44.759.294. He seleccionado como lenguaje de programación Javascript, inspirado en una implementación de redes neuronales convolucionales escrita en este lenguaje (“ConvNetJS: Deep Learning in your browser,” 2017), la experiencia es buena, pero siento que con Java o C++, incluso C#, hubiese logrado resultados más óptimos, sin embargo, escribir este tipo de algoritmos en un lenguaje no tipico me sirvió bastante para comprender un poco más acerca de este paradigma.

Algoritmo genético

Para empezar la implementación de un algoritmo genético básico consulté los apuntes de la asignatura (Berzal, 2017), sin embargo, también es necesario un mecanismo de selección y de reemplazo, un operador de cruce y de mutación. También fue muy útil realizar una revisión de distintas implementaciones presentes en repositorios públicos (“Build software better, together,” 2017) y una entrada en un blog (“Algoritmos Genéticos (AGs),” 2008).

Mecanismo de selección (selección de padres)

Selección proporcional ruleta y selección por torneo, un híbrido entre la probabilidad y la aleatoriedad, en búsqueda del mejor.

Mecanismo de reemplazo (selección de supervivientes)

Elitismo, donde los mejores individuos se pasan directamente a la siguiente generación, tal como se menciona en clase y en los apuntes (Berzal, 2017).

Operador de cruce

Se busco generar la solución de cruzar dos cromosomas, referentes a dos puntos de corte, generando dos hijos diferentes, uno con más prioridad hacia el padre que la madre y el otro hacia la madre que el padre.

Operador de mutación

Intercambio de alelos, genes que intercambiaran al azar sus alelos, donde existe una probabilidad de seleccionar un gen para que sufra una mutación.

Optimización local

Algoritmo 2-opt, realiza una optimización local de un cromosoma utilizando un algoritmo greedy basado en 2-opt para la variantes del algoritmo genético.

Variantes

Tomando como base los operadores y mecanismos definidos anteriormente se agregaran técnicas de optimización local, para que la población “aprenda”, inspirados en Baldwin, y que además de aprender, los descendientes hereden lo aprendido, inspirados en Lamarck (Berzal, 2017).

Experimentación

En la Tabla 1 se pueden los valores de parámetros utilizados.

Algoritmo	B-Básico, W-Baldwin, L-Lamarck
Tamaño de la población	de 10 a 100
Probabilidad de mutación	mayor a *n*, donde *n* va de 0.05 a 0.5
Probabilidad de optimización	mayor a *n*, donde *n* va de 0.7 a 0.99
Porcentaje de población	de 15 a 70%

Tabla 1. Valores de parámetros utilizados a lo largo de la experimentación.

A lo largo de la experimentación fueron probándose parámetros y ajustando los algoritmos, tanto para lograr un buen resultado, como para lograr un rendimiento eficiente y aceptable.

Ocurría en muchos casos que le navegador quedaba congelado: Internet Explorer 11, Chrome 55, presentaron mejores prestaciones para la ejecución de este algoritmo, mientras que el menos apto resultó ser Mozilla Firefox 51, incluso por debajo del novel Microsoft Edge 38.1. En la Figura 1 se puede ver el mejor fitness obtenido con el algoritmo estándar: 49.580.608, en la Figura 2, con la variante inspirada en Baldwin: 44.918.228, y en la Figura 3, en Lamarck: 44.878.390.

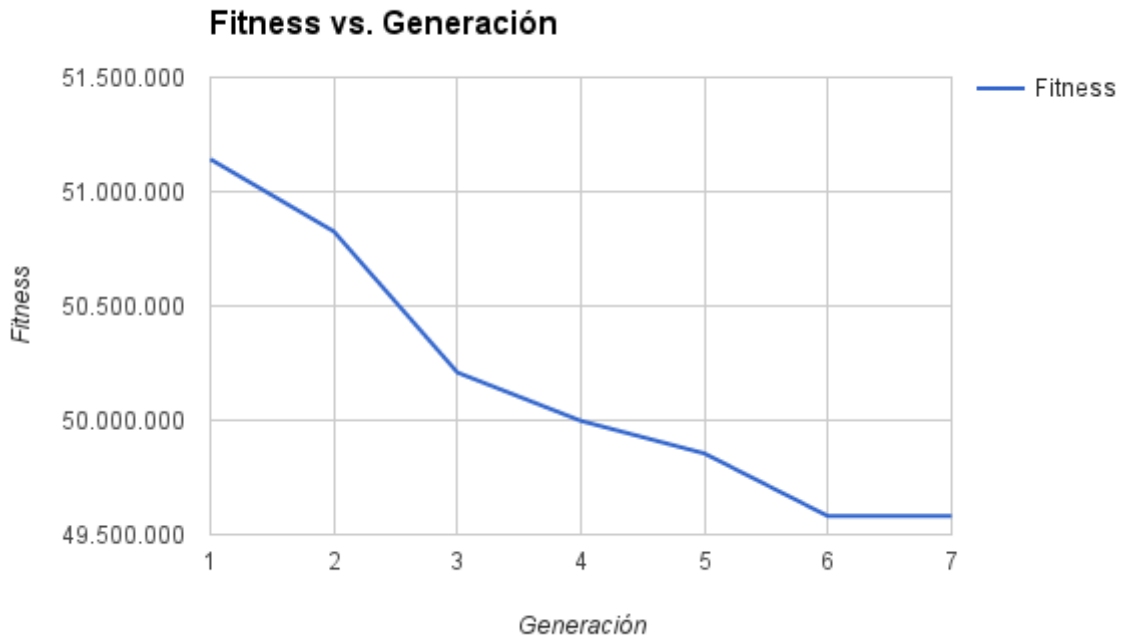

Figura 1. Fitness obtenido en generaciones con el algoritmo básico.

Referente a los valores de los parámetros, con los que mejores resultados se han obtenido en cuanto a fitness son:

- Algoritmo: Lamarck,
- Población: 40
- Probabilidad de mutar: 0.2,
- Porcentaje de población: 70%,
- Porcentaje optimización a población: 0 (100%)

Si evaluamos los tiempos de ejecución, hay que tener en cuenta que ralentiza mucho al ordenador que lo ejecuta y se debe indicar al navegador del mismo, que confíe y espere a que termine la ejecución del script, sin embargo con estos valores se realiza una ejecución razonable:

- Algoritmo:
    - Básico,
        - Población: hasta 6000 (lo que se pudo probar),
    - Variantes:
        - Población: hasta 40,
- Probabilidad de mutar: 0.3,
- Porcentaje de población: 10%,
- Porcentaje optimización a población: 0.99 (1%).

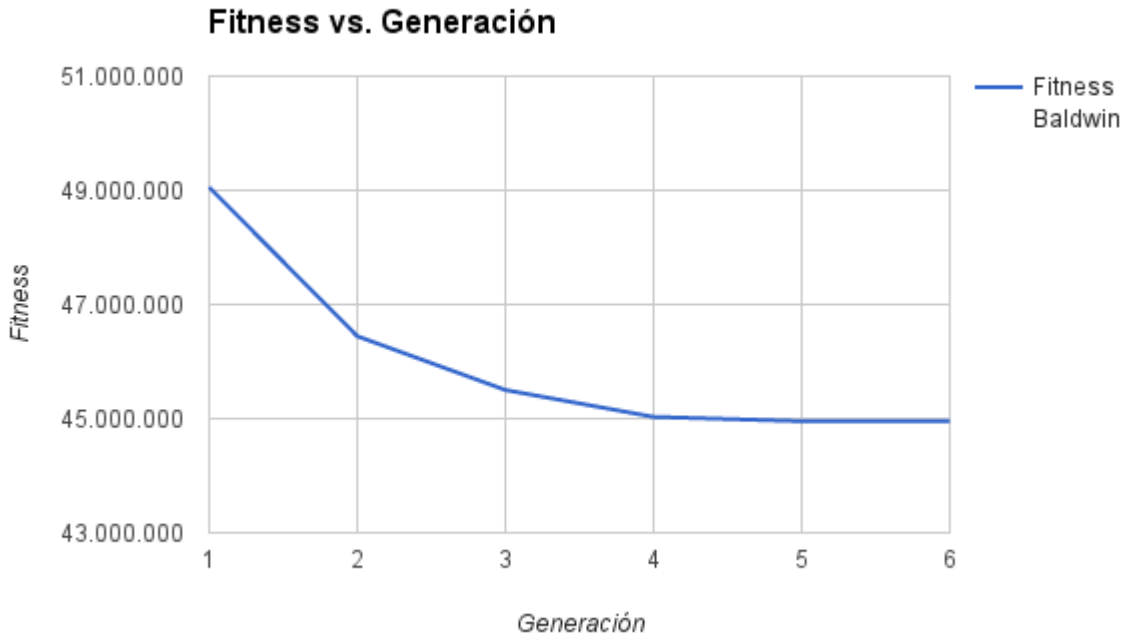

Figura 2. Fitness obtenido en generaciones con el algoritmo inspirado en Baldwin.

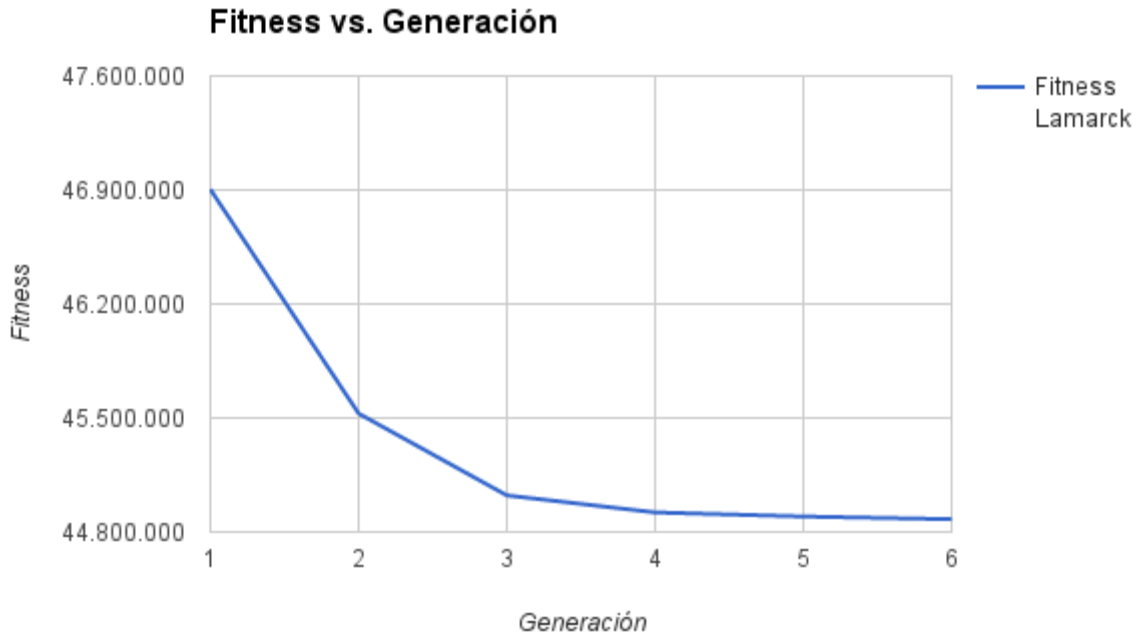

Figura 4. Fitness obtenido en generaciones con el algoritmo inspirado en Lamarck.

Análisis de resultados

En la Figura 4 se ilustran los resultados de fitness obtenidos contra la población utilizada para cada algoritmo. De izquierda a derecha se observa los que obtienen mejores resultados, son los algoritmos que utilizan variantes, los de Baldwin y Lamarck, por sobre el “básico”.

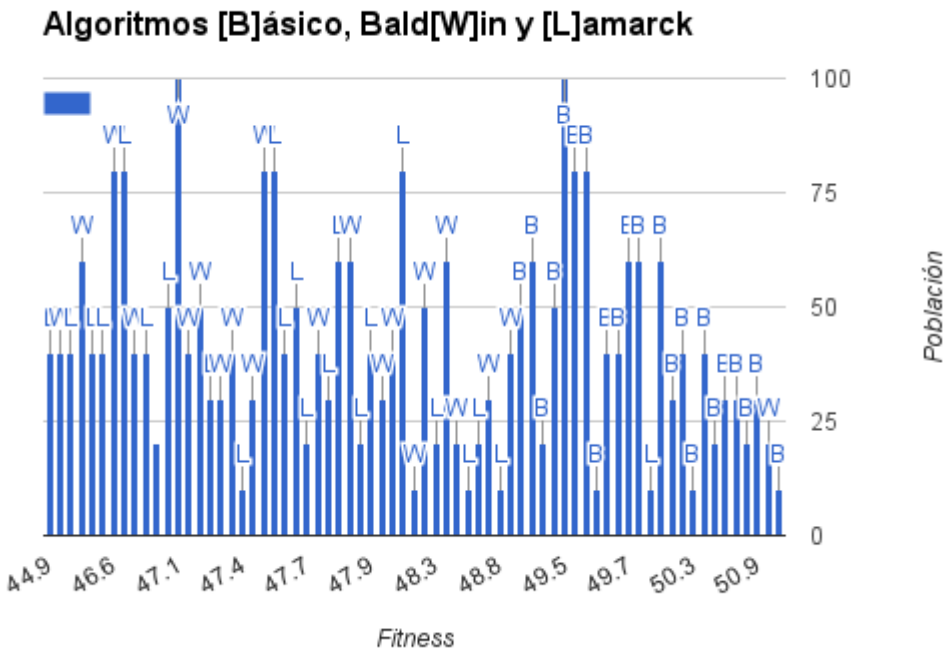

Figura 4. Resultados obtenidos por los algoritmos en la experimentación.

Tras las experimentaciones se observa que resulta más eficiente aplicar una optimización inspirada en Baldwin tan solo a una parte de la población, en lugar de a toda, por términos de tiempo de ejecución y de resultados, ya que se incrementa la variedad de la población, punto importante para huir de “óptimos locales”. Esto también se aplica a la optimización basada en Lamarck.

En las Figura 4 se observan todas las variantes aplicadas del algoritmo genético, siendo las variables optimizadas, las superiores en todos los casos, inclusive desde las primeras generaciones contra el algoritmo estándar. Entre las variantes, las inspiradas en Lamarck contra las inspiradas en Baldwin, obtienen mejores resultados (ver Figura 5). Cabe destacar, en cuanto al tiempo, la ejecución de estas variantes optimizadas necesitan más tiempo debido al algoritmo 2-opt, sobresaliendo ampliamente sobre estas el algoritmo estándar, que necesita un menor tiempo de ejecución. Otro punto que se puede agregar, que con una población de 40 a 50 individuos ya pude lograr mi mejor resultado, en un máximo de siete generaciones.

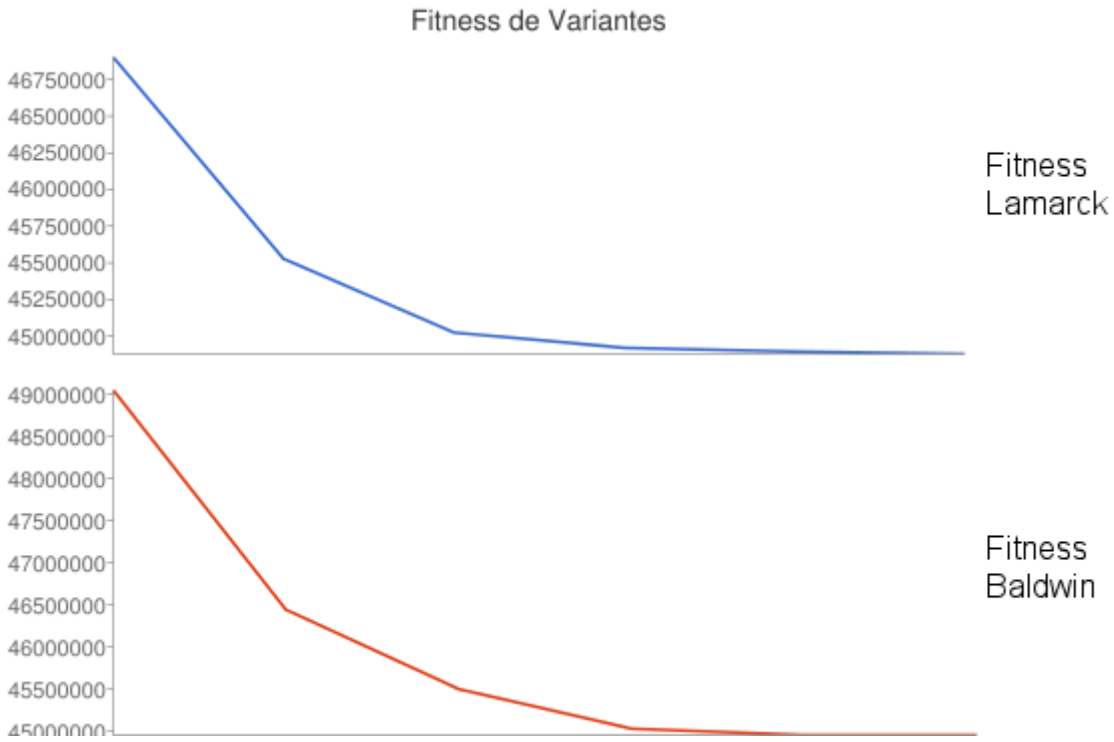

Figura 5. Comparación entre variantes del algoritmo genético estándar.

Conclusiones

Podemos concluir que los algoritmos genéticos son no deterministas, donde el azar juega un papel importante, por lo que en las diferentes ejecuciones difieren en los resultados obtenidos. Sobre las variantes optimizadas de los algoritmos no se ha encontrado demasiada diferencia, sobresaliendo la lamarckiana, aunque parece depende mucho el azar y la implementación realizada.

En mi caso particular, al utilizar Javascript, un lenguaje ejecutado del lado del cliente, no pude acercarme mucho al mejor resultado, incluso aplicando la orientación a objetos, quedandome en un modesto 44.878.390, que de haber utilizado otro lenguaje ejecutado del lado del servidor o compilado, podría haberse mejorado.

Bibliografía

Algoritmos Genéticos (AGs). (2008, December 4). Retrieved January 28, 2017, from https://just4cool.wordpress.com/2008/12/04/algoritmos-geneticos-ags/

Berzal, F. (2017). Curso de Inteligencia Computacional. Retrieved January 27, 2017, from http://elvex.ugr.es/decsai/computational-intelligence/

Build software better, together. (2017). Retrieved January 27, 2017, from https://github.com/search?utf8=%E2%9C%93&q=qap

Burkard, R. E., Çela, E., Karisch, S. E., & Rendl, F. (1997). QAPLIB Home Page. Retrieved January 27, 2017, from http://anjos.mgi.polymtl.ca/qaplib//

ConvNetJS: Deep Learning in your browser. (2017). Retrieved January 27, 2017, from http://cs.stanford.edu/people/karpathy/convnetjs/