

Máster Universitario en Ingeniería Informática

INTELIGENCIA COMPUTACIONAL

PRÁCTICA 2: Algoritmos Evolutivos QAP



**UNIVERSIDAD
DE GRANADA**



Carlos Santiago Sánchez Muñoz

Grupo de prácticas 1 - Lunes

Email: carlossamu7@correo.ugr.es

4 de enero de 2020

Índice

1. Descripción del problema	2
2. Operadores	3
2.1. Operador de cruce	3
2.2. Operador de mutación	3
2.3. Operador de selección	3
3. Algoritmos	4
3.1. Algoritmo Genético Generacional	4
3.2. Algoritmo Genético Estacionario	4
3.3. Variante Baldwiniana	4
3.4. Variante Lamarckiana	5
4. Comparativa	6
4.1. AGE	6
4.2. AGG	9
5. Resultados	12

1. Descripción del problema

El problema de la asignación cuadrática o QAP [Quadratic Assignment Problem] es un problema de optimización combinatoria con numerosas aplicaciones. El problema se puede describir de la siguiente forma.

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

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

donde $p()$ define una permutación sobre el conjunto de instalaciones.

Igual que en el problema del viajante de comercio, que se puede considerar un caso particular del QAP, una solución para el problema es una permutación del conjunto de instalaciones que indica dónde se debe construir cada una.

El problema de la asignación cuadrática es un problema habitual en Investigación Operativa y, además de utilizarse para decidir la ubicación de plantas de producción, también se puede utilizar como modelo para colocar los componentes electrónicos de un circuito sobre una placa impresa o los módulos de un circuito integrado en la superficie de un microchip.

Por su interés teórico y práctico, existe una variedad muy amplia de algoritmos que abordan la resolución del problema de la asignación cuadrática. Al ser un problema NP-completo, el diseño y aplicación de algoritmos exactos para su resolución no es viable cuando n es grande. Nos centraremos, por tanto, en el diseño de algoritmos evolutivos y evaluaremos su rendimiento sobre instancias concretas del problema.

2. Operadores

Los algoritmos genéticos son algoritmos bioinspirados basados en la evolución. Como tales tienen operadores de cruce, mutación y selección. En algunos casos hay varios de ellos. A lo largo de este trabajo se realizará una comparativa de como tomar una decisión u otra puede afectar a los resultados.

2.1. Operador de cruce

Estos operadores deciden cómo se construyen los hijos de una nueva generación. Es decir, como cruzar dos padres para obtener los hijos.

El primer operador que se ha implementado es el **cruce OX** en donde se escoge una subcadena de uno de los padres y la subcadena complementaria del otro padre completando el hijo. En la elección de esta subcadena he usado los dos cuartos centrales del primer ascendiente y completo con el primer y último cuarto del otro ascendiente.

El otro operador es el de **cruce de posición** que mantiene en los descendientes aquellos genes comunes y el resto eligiendo de forma aleatoria.

2.2. Operador de mutación

Al igual que en la genética biológica también se implementan operadores de mutación que permite alterar la población. Esto resulta muy útil ya que permite la aparición de nuevas soluciones y muchas veces permite salir de óptimos locales para viajar a mejores soluciones.

El operador de mutación implementado es tan sencillo como intercambiar dos genes cualesquiera.

2.3. Operador de selección

El otro ingrediente fundamental de los algoritmos evolutivos es el operador de selección. Éste decide qué elementos de la población continúan en la siguiente generación. Se ha implementado un **torneo binario**. Dos individuos de la población son escogidos aleatoriamente y compiten entre ellos mediante su coste o *fitness*. He ordenado la población de modo que en el torneo binario el mejor es aquel que tiene un índice inferior. De este modo los padres son los mejores posibles manteniendo el elitismo.

3. Algoritmos

En esta sección se va a contar brevemente los algoritmos implementados. Antes de entrar en los tecnicismo de los algoritmos hay que decidir cómo establecer la población inicial. Hacerlo de forma aleatoria sin repetición es claramente una posibilidad.

¿Existe alguna forma mejor? La respuesta es afirmativa. Por ejemplo podemos usar como población inicial una generada mediante un **algoritmo greedy**. Dicho algoritmo busca minimizar la distancia a la vez que maximizar el flujo ya que es un problema de optimización múltiple. El tamaño de la población elegido es 20.

3.1. Algoritmo Genético Generacional

Este algoritmo en cada generación sustituye la población actual por la nueva. Se escogen los padres mediante el torneo binario y existe una **probabilidad de cruce** de que estos se reproduzcan. Los padres no cruzados se mantienen de modo que la población de una iteración con la de la siguiente tenga la misma dimensión.

Asimismo también se usa el operador de mutación que permite intercambiar dos genes en función de una **probabilidad de mutación**. El último componente esencial de estos algoritmos es que el mejor individuo de la población ha de sobrevivir. Esto se conoce como **elitismo**.

Tras realizar pruebas los mejores parámetros hallados son:

Probabilidad de cruce: 0.7

Probabilidad de mutación. 0.001

3.2. Algoritmo Genético Estacionario

Esta otra modalidad tiene diferencias notables respecto a la anterior. En cada iteración se escogen dos padres (aleatoriamente) y se les aplica los operadores genéticos. Del mismo modo sólo se incluyen los hijos si son mejores que las dos peores soluciones de la generación en cuestión.

Lógicamente este algoritmo toma menos tiempo que el Generacional y a veces permite mantener soluciones que a la postre nos permiten llegar a soluciones muy buenas.

3.3. Variante Baldwiniana

Esta variante incorpora técnicas de optimización local al algoritmo genético estándar para dotar a los individuos de aprendizaje. Se evalúa el fitness teniendo en cuenta esta optimización pero los individuos a usar son con el material genético original. Se ha implementado el **algoritmo greedy de transposición 2-opt** como técnica de optimización.

Obsérvese un pequeño fragmento de código, que permite entender muy bien esta idea:

```
void Geneticos::Baldwiniana(vector<Cromosoma> &pop) {  
    for (unsigned i=0; i<pop.size(); ++i) {  
        algGreedy.Alg2opt(pop[i], datos);  
        pop[i].fitness = algGreedy.solutionGreedy.fitness;  
    }  
}
```

3.4. Variante Lamarckiana

Esta variante es análoga a la anterior pero manteniendo el material genético aprendido en el proceso de aprendizaje. A nivel de código:

```
void Geneticos::Lamarckiana(vector<Cromosoma> &pop) {  
    for (unsigned i=0; i<pop.size(); ++i) {  
        algGreedy.Alg2opt(pop[i], datos);  
        pop[i].solution = algGreedy.solutionGreedy.solution;  
        pop[i].fitness = algGreedy.solutionGreedy.fitness;  
    }  
}
```

4. Comparativa

Para dicha comparativa se ha usado el conjunto de datos `tai60a.dat`.

4.1. AGE

Comenzamos con los Algoritmos Estacionarios.

Algoritmo estándar con población inicial greedy

Tabla de resultados:

IterGenetic	FitnessGenetic	FitnessGenetic OX
20000	7637546	7727468
40000	7523292	7536962
60000	7523292	7503284
80000	7523292	7503284
100000	7523292	7503284
120000	7523292	7503284
140000	7523292	7503284
160000	7523292	7503284
180000	7523292	7503284
200000	7523292	7503284

Gráfica:

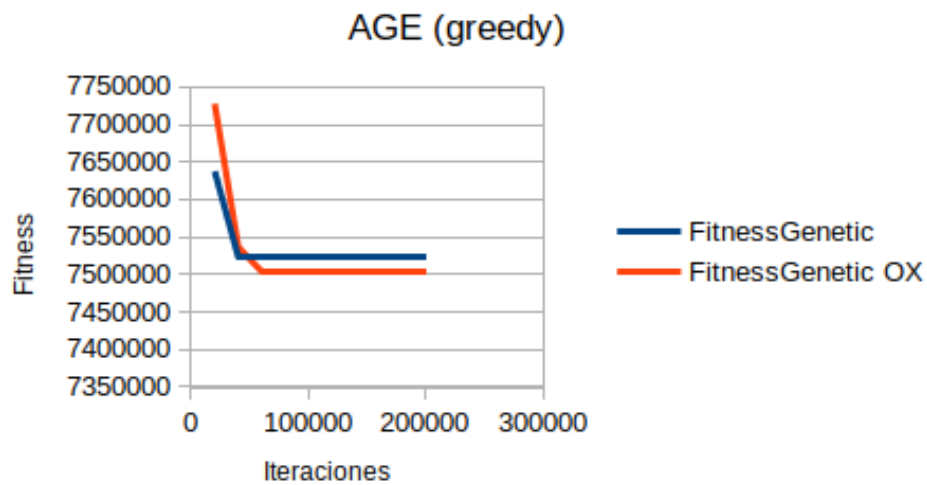


Imagen 1: Algoritmo estándar con población inicial greedy (AGE)

Comienza desde un fitness decente debido a su inicialización greedy y decae rapidísimamente para estacionarse en un óptimo local. El cruce OX es algo superior.

Baldwiniana

Tabla de resultados:

IterBaldwiniana	FitnessBaldwiniana	FitnessBaldwiniana OX
10000	7641366	7727526
20000	7598224	7617646
30000	7598224	7593314
40000	7598224	7575026
50000	7598224	7557268
60000	7598224	7557268
70000	7598224	7557268
80000	7598224	7557268
90000	7598224	7557268
100000	7598224	7557268

Gráfica:

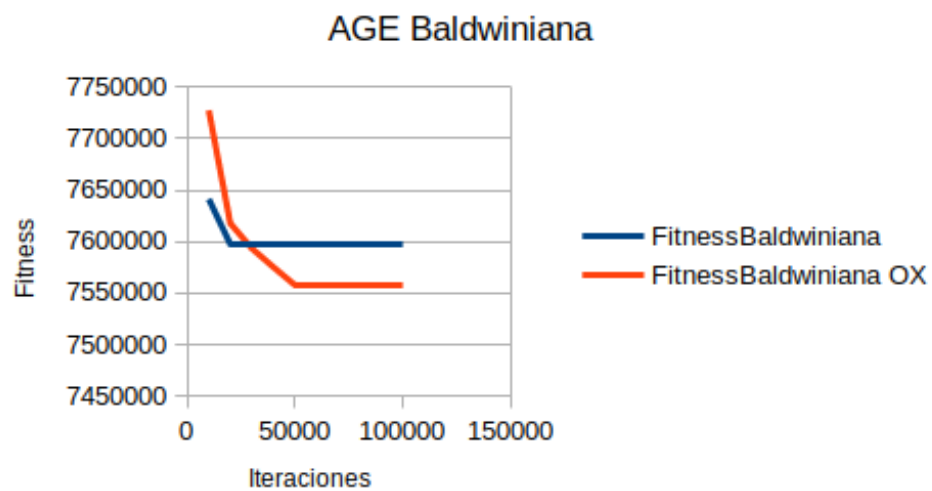


Imagen 2: Baldwiniana (AGE)

En este caso se observa que la mejora del *fitness* es más progresiva con una buena pendiente de decrecimiento. Posteriormente también estaciona. El cruce OX es bastante superior al de posición.

Lamarckiana

Tabla de resultados:

IterLamarckiana	FitnessLamarckiana	FitnessLamarckiana OX
1000	7560410	7560410
2000	7560410	7560410
3000	7560410	7560410
4000	7560410	7560410
5000	7560410	7560410
6000	7560410	7560410
7000	7560410	7560410
8000	7560410	7560410
9000	7560410	7560410
10000	7560410	7560410

Gráfica:

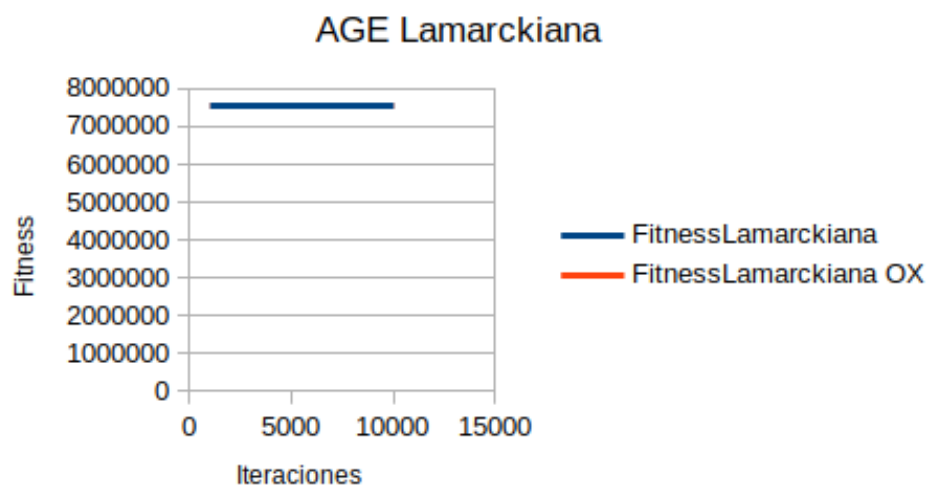


Imagen 3: Lamarckiniana (AGE)

No existe variación alguna entre usar 1000 iteraciones o 10000, ambas proveen el mismo *fitness*. Quizás se debería probar con valores más pequeños como trabajo futuro pero el hecho de que el aprendizaje permanezca entre iteraciones genera este efecto, al menos sobre este conjunto de datos.

4.2. AGG

Ahora pasamos a comparar resultados en los Algoritmos Generacionales.

Algoritmo estándar con población inicial greedy

Tabla de resultados:

IterGenetic	FitnessGenetic	FitnessGenetic OX
20000	7776278	7832018
40000	7725174	7745818
60000	7682876	7646762
80000	7670010	7631202
100000	7640942	7584130
120000	7619880	7583892
140000	7607828	7580972
160000	7587250	7569986
180000	7587250	7534020
200000	7583398	7534020

Gráfica:

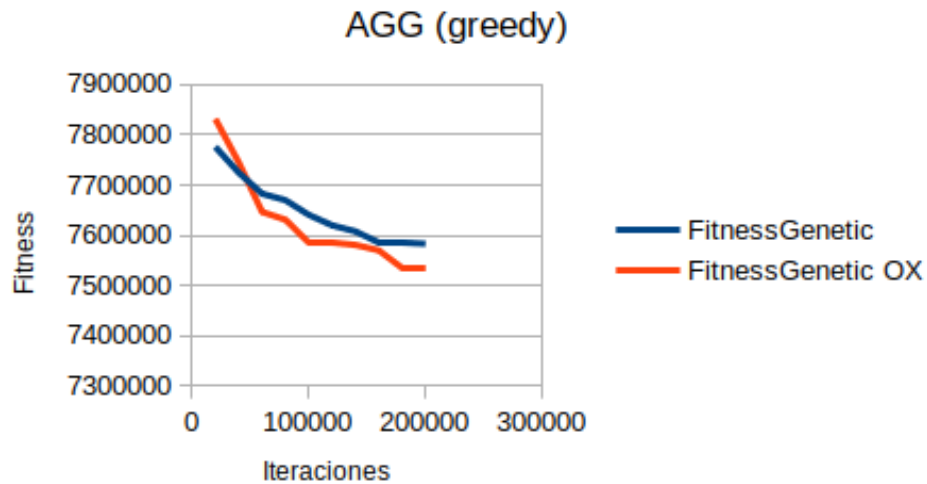


Imagen 4: Algoritmo estándar con población inicial greedy (AGG)

En este caso la caída es muy diferente a lo que ya conocíamos. El cambio de población es más brusco aumentando la complejidad computacional pero también se puede mejorar el coste. Parece que en las últimas iteraciones ya ha estacionado. Como hasta ahora, el cruce OX es superior.

Baldwiniana

Tabla de resultados:

IterBaldwiniana	FitnessBaldwiniana	FitnessBaldwiniana OX
10000	7899320	8075532
20000	7717128	7821946
30000	7760356	7852368
40000	7630506	7795604
50000	7663012	7695824
60000	7622826	7712322
70000	7634528	7722406
80000	7634528	7722406
90000	7567130	7722128
100000	7619172	7697428

Gráfica:

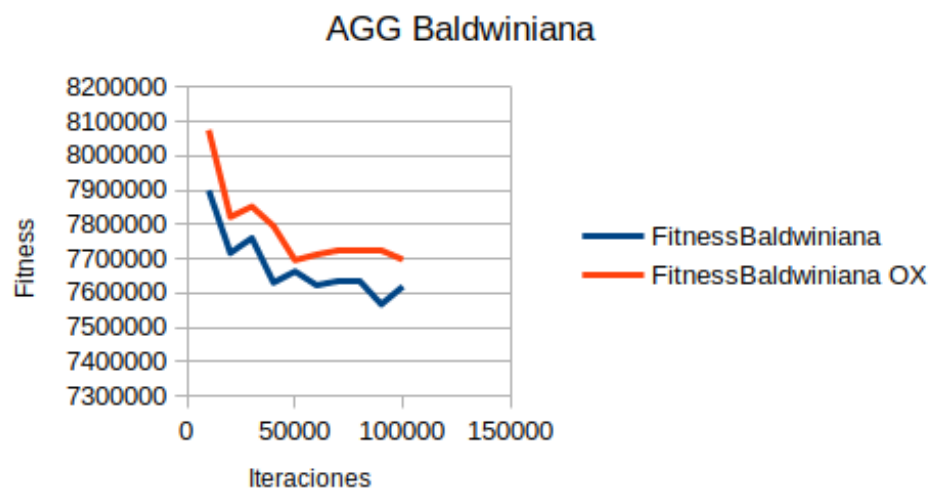


Imagen 5: Baldwiniana (AGG)

Se observan oscilaciones y a veces un aumento de iteraciones puede llevar a un *fitness* peor pero a la larga las soluciones son mejores. En este caso el operador de cruce de posición es mejor. Las mutaciones y cambios drásticos en la población explican dichas leves oscilaciones.

Lamarckiana

Tabla de resultados:

IterLamarckiana	FitnessLamarckiana	FitnessLamarckiana OX
1000	7947652	7947652
2000	7607836	7693590
3000	7542780	7630272
4000	7542780	7630272
5000	7542780	7630272
6000	7542780	7630272
7000	7542780	7630272
8000	7542780	7630272
9000	7542780	7630272
10000	7542780	7630272

Gráfica:

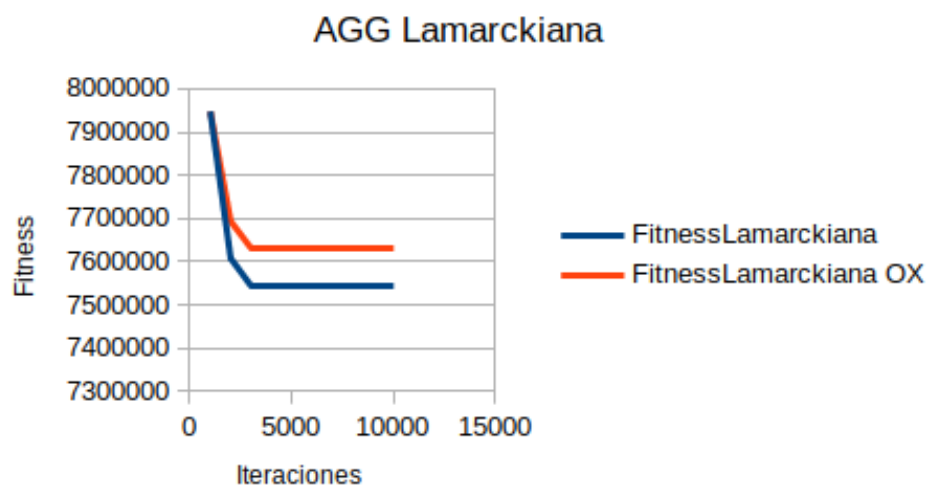


Imagen 6: Lamarckiniana (AGG)

Como era de esperar el coste cae rápidamente para establecerse en un óptimo. Al igual que el caso anterior el cruce por posición es mejor al OX.

5. Resultados

Tras los resultados obtenidos en la comparativa anterior los diferentes algoritmos y variantes genéticos no ofrecen grandes diferencia en los resultados siendo todas ellas bastante buenas. Asimismo pasa con el operador de cruce, no hay evidencia para pensar que OX sea mejor que posición o viceversa.

En esta sección se va a ofrecer los resultados sobre el conjunto de datos pedido, `tai256.c`, el cual tiene un número de instalaciones considerablemente alto.

Mejor *fitness*: 45133724.

Solución: 132 98 130 13 96 215 49 81 94 229 178 167 125 73 187 246 127 170 244 115 64 207 22 26 46 37 69 60 7 120 24 202 67 43 20 71 161 39 17 9 190 252 175 204 3 84 113 122 135 224 34 221 0 184 250 117 77 63 238 90 212 107 155 226 195 209 173 180 144 255 217 31 28 137 56 235 147 142 152 140 241 165 198 58 103 86 105 52 176 150 92 232 181 210 240 237 216 76 136 251 211 101 201 121 126 197 53 220 40 19 233 93 247 139 68 234 6 25 23 78 182 151 141 248 206 21 205 189 111 171 8 168 208 41 1 118 188 48 242 102 163 61 162 30 74 10 70 79 50 166 42 62 87 199 156 253 82 119 124 179 218 138 44 51 104 91 133 134 4 254 146 55 231 225 100 145 236 157 174 59 249 2 83 36 185 85 75 33 18 196 11 172 245 89 45 239 154 32 57 192 219 223 230 203 243 128 227 213 97 12 65 106 160 169 191 143 159 123 95 29 222 27 54 47 116 186 16 109 194 80 110 66 108 14 158 99 228 131 164 35 153 15 129 72 38 200 193 112 177 114 5 148 88 183 214 149.

Resultados de AGE:

```
./bin/age -m -s -p -f /home/.../tai256c.dat
-----
Intelligence Computational GAT
-----
----- AGE P (Initial greedy) -----
Iteraciones: 150000
Tiempo: 39.8725s
Solución:
23 126 253 243 4 60 103 159 61 232 152 224 9 35 128 197 26 169 189 238 21 18 247 48 209 176 282 92 94 172 174 150 182 223 235 204 131 83 88 221 46 90 33 200 120 217 194 181 241 250 183 98 71 179 155 0 138 111 57 187 31 80 146 164 6
12 189 135 113 101 226 75 133 187 116 214 185 79 148 49 11 212 40 229 81 280 157 38 28 14 235 43 285 129 162 93 72 17 287 13 213 144 37 239 163 78 137 18 44 124 162 220 69 19 13 168 96 85 166 227 230 56 173 143 14 47 1 82 218 184
104 251 123 193 49 36 32 63 139 16 108 192 73 15 152 42 177 22 147 119 231 248 160 25 112 186 134 225 165 117 95 64 198 15 118 67 215 59 141 82 77 213 195 114 249 125 7 151 121 87 178 12 201 30 74 244 84 190 1 142 122 171 97 199 16
6 237 68 252 51 3 45 130 50 240 190 80 240 136 180 180 148 228 110 70 181 222 99 41 185 210 283 236 39 134 234 219 178 167 65 68 91 153 27 158 150 191 175 8 254 242 149 211 188 115 127 145 20 210 24 29 54 58 2 245 208
Fitness: 4505732
-----
----- AGE OR (Initial greedy) -----
Iteraciones: 150000
Tiempo: 38.6472s
Solución:
39 148 191 11 230 21 9 125 2 101 220 160 119 180 138 159 80 289 54 0 224 87 94 41 228 35 3 235 254 150 75 245 142 17 107 179 194 72 215 45 13 97 32 184 140 237 114 58 15 00 69 280 20 24 30 247 250 283 171 197 52 241 152 127 211 192
173 98 222 218 77 165 92 121 131 117 285 123 63 140 188 182 169 58 67 162 82 129 233 180 28 65 5 130 124 59 149 109 107 229 99 154 190 36 120 190 158 122 40 238 51 93 240 189 43 234 88 25 23 168 34 212 251 248 218 249 227 95 181 1
35 184 67 208 140 1 118 243 64 242 89 147 217 156 198 74 151 78 172 175 160 83 78 65 185 158 253 163 111 143 80 235 284 44 134 19 141 81 170 43 239 144 55 211 275 64 145 238 178 182 232 137 4 115 30 22 47 110 126 33 18 79 37 195 16
4 12 219 168 98 213 133 180 8 113 132 61 174 31 27 139 181 223 18 38 177 78 73 49 178 128 187 281 6 244 16 48 110 162 135 81 165 183 282 220 14 221 153 252 71 56 287 169 216 66 214 62 7 90 183 186 46 68 157 53 29 193 286
Fitness: 45133724
-----
----- AGE P Baldwiniano -----
Iteraciones: 50008
Tiempo: 10.1962s
Solución:
299 148 124 177 58 253 149 163 223 165 216 5 7 138 68 134 90 90 2 69 128 99 92 236 86 159 219 94 171 54 151 88 220 199 190 56 37 231 46 192 43 34 82 51 63 120 39 77 131 248 49 289 162 155 12 29 173 184 14 140 233 18 240 81 16 182 1
9 127 73 160 116 285 236 212 197 41 114 123 282 169 79 226 179 22 194 84 185 251 243 188 151 240 232 132 249 78 234 21 24 150 119 247 280 126 44 186 158 45 112 250 224 252 0 13 189 242 66 11 38 208 117 140 70 53 200 180 15 201 235
111 1 183 93 198 121 211 83 162 187 178 48 239 218 130 160 180 17 47 181 25 188 287 89 118 82 27 61 115 190 133 127 118 184 239 113 213 67 74 8 172 32 170 221 9 178 125 228 175 167 284 174 188 185 76 73 225 133 50 147 33 193 98 71
68 245 68 28 181 30 214 28 129 35 42 153 139 215 180 72 217 157 31 143 161 218 238 3 28 154 145 154 237 158 80 148 122 55 46 62 65 195 254 255 130 142 181 157 97 18 244 183 1 50 57 93 144 23 283 6 241 87 32 30 187 95 222
Fitness: 45498096
-----
----- AGE OR Baldwiniano -----
Iteraciones: 50000
Tiempo: 9.5213s
Solución:
119 191 112 6 100 9 183 170 67 130 180 222 178 159 110 43 20 215 132 90 200 95 155 20 97 195 48 60 3 0 149 14 98 37 87 221 213 18 72 77 45 65 227 219 23 250 190 62 55 244 31 217 125 185 33 145 70 40 224 248 12 117 79 193 283 192 17
1 181 123 145 78 85 188 187 90 235 163 142 152 140 241 186 235 28 233 92 169 52 240 150 188 182 5 229 189 180 16 159 136 37 99 83 187 30 120 205 180 219 49 194 213 225 28 89 69 197 212 25 184 86 172 151 54 63 223 22 224 126 32 171
24 176 280 71 1 118 134 64 242 183 147 7 161 169 74 19 47 168 141 160 42 80 228 21 158 232 81 111 96 83 182 11 44 51 104 91 144 110 4 239 122 245 178 30 84 140 238 158 174 154 249 121 218 131 179 231 31 120 181 88 78 114 19 289
165 133 214 8 167 252 61 15 109 153 17 104 2 38 230 243 211 148 41 129 187 201 251 27 180 240 78 162 34 135 73 177 46 82 254 284 139 226 237 238 94 287 13 216 66 247 113 220 143 56 282 127 68 59 157 53 138 190 280 39 128
Fitness: 45498096
-----
----- AGE P Lamarckiano -----
Iteraciones: 18800
Tiempo: 3483.24s
Solución:
37 132 130 13 90 215 98 81 94 229 252 184 125 128 187 246 127 22 244 115 64 287 22 26 46 24 69 68 7 170 167 282 67 26 20 71 161 39 43 9 190 6 175 284 3 84 113 122 135 289 34 221 56 178 250 117 77 63 238 90 212 187 155 226 195 224 1
73 180 144 255 217 31 28 137 73 235 147 142 152 140 241 165 198 58 183 80 185 52 176 158 92 232 181 218 246 179 216 73 130 251 211 181 281 121 120 197 53 220 41 19 233 93 247 139 68 234 6 25 23 78 182 151 141 248 286 21 281 189 111
171 8 168 288 194 1 118 188 48 242 182 163 61 162 36 74 19 79 58 160 42 62 87 199 158 233 82 109 124 185 218 138 44 51 184 91 133 134 4 18 146 35 231 225 180 143 286 118 174 58 249 2 83 38 280 65 75 33 68 180 254 172 245 89 45
239 154 21 48 193 219 223 238 283 243 128 227 213 97 12 65 180 160 157 191 143 119 123 95 29 222 27 54 47 116 186 16 169 32 80 110 66 188 14 158 99 228 131 164 35 153 15 129 72 38 192 11 112 177 114 5 148 57 183 214 149
Fitness: 44919626
-----
----- AGE OR Lamarckiano -----
Iteraciones: 1880
Tiempo: 4879.04s
Solución:
132 98 130 13 96 215 49 81 94 229 178 167 125 73 187 246 127 170 244 115 64 207 22 26 46 37 69 60 7 120 24 202 67 43 20 71 161 39 17 9 190 252 175 204 3 84 113 122 135 224 34 221 0 184 250 117 77 63 238 90 212 107 155 226 195 209 1
73 180 144 255 217 31 28 137 56 235 147 142 152 140 241 165 198 58 183 80 185 52 176 158 92 232 181 218 246 179 216 73 130 251 211 181 281 121 120 197 53 220 41 19 233 93 247 139 68 234 6 25 23 78 182 151 141 248 286 21 281 189 111
171 8 168 288 194 1 118 188 48 242 182 163 61 162 36 74 19 79 58 160 42 62 87 199 158 233 82 119 124 185 218 138 44 51 184 91 133 134 4 254 146 55 231 225 180 145 236 118 174 58 249 2 83 38 185 85 75 33 18 196 11 172 245 89 45 2
88 144 32 57 182 219 223 238 283 243 128 227 213 97 12 65 180 160 169 131 143 119 123 95 29 222 27 54 47 116 186 16 169 184 68 118 66 188 14 158 99 228 131 164 35 153 15 129 72 38 280 193 112 177 114 5 148 88 183 214 149
Fitness: 44919626
```

Imagen 7: Resultados AGE sobre `tai256c.dat`