**Nombre:** Guillermo García Arredondo

**DNI:** 75927655D

**Curso académico:** 3º

**E-mail:** ggarredondo@correo.ugr.es

# Práctica Alternativa – Optimización Muscular

# Índice

# Parte I. Descripción de la Optimización Muscular.

## Introducción a la idea

La denominada *optimización muscular* es una metaheurística original de mi autoría que se origina como contrapartida de los algoritmos genéticos. Si entendemos los AGs como algoritmos que reflejan el fenómeno donde una población/especie se adapta al entorno, la optimización muscular pretende reflejar el fenómeno donde el individuo se adapta a la tarea que tiene que resolver, desarrollando aquellas cualidades que dicha tarea requiere de él.

Desde el punto de vista natural, estas cualidades incluirían atributos característicos de la musculatura como fuerza o resistencia (de ahí la denominación), pero no se limita a éstas pues los seres vivos no dependemos únicamente de nuestra corpulencia para sobrevivir. El objetivo *no* es, por tanto, plasmar el complejo funcionamiento de reconstrucción y fortalecimiento de los músculos, sino reflejar la capacidad general de los animales (y en especial de los seres humanos) de mejorar sus aptitudes con tal de desempeñar eficientemente una tarea.

## ¿Cuál es la tarea?

Tal pregunta puede parecer trivial, ya que la tarea final siempre será alcanzar el mejor fitness. No obstante, es relevante extenderse un poco pues ahí yace la principal particularidad del algoritmo.

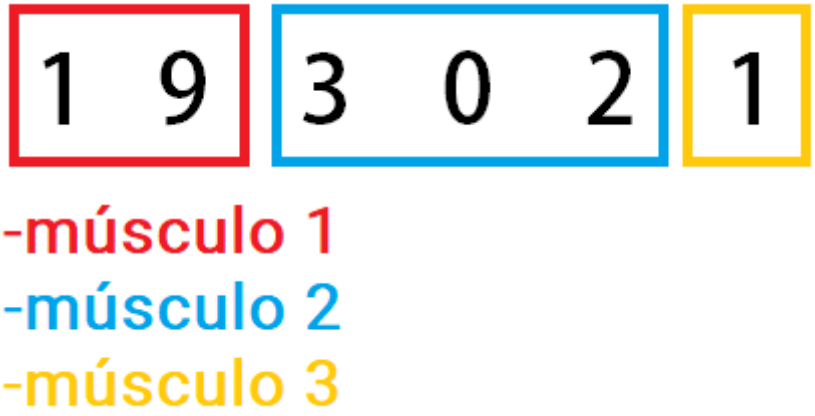
Volviendo a la analogía con la naturaleza, situándonos concretamente en el caso del hombre prehistórico, podríamos decir que su principal tarea es sobrevivir (y reproducirse, pero lo obviaremos por simplificar el ejemplo). Este objetivo, sin embargo, se puede lograr desde numerosos enfoques.

Para sobrevivir necesitará comida, y para comer necesitará cazar. Para llevar a cabo esta subtarea, podría desarrollar una musculatura que le permitiría correr más deprisa y no perder a su presa. También podría desarrollar la capacidad cardiaca y pulmonar (resistencia) para poder mantener esa carrera durante largas distancias. Incluso, podría desarrollar la inteligencia emocional necesaria para liderar un grupo de cazadores y coordinar un ataque que le asegurase atrapar a la presa. Podría desarrollar cualquiera de estas habilidades, o todas a la vez.

En resumen, el hombre prehistórico se entrenará en diversas aptitudes que en conjunto le harán un individuo más capaz de llevar a cabo la tarea que es sobrevivir. Ahí es donde surge el verdadero funcionamiento de la metaheurística: si suponemos

la solución como un vector de elementos, la *optimización muscular* dividirá esta solución en un número determinado de segmentos de tamaño aleatorio que serán las aptitudes (o como lo llamaremos a partir de ahora, músculos), e irá seleccionando y mejorando dichos segmentos siguiendo unos *criterios* (que representarían las subtareas).

Ejemplo de la representación:




Por lo tanto, el algoritmo resultante será uno de explotación que se centrará en mejorar y hacer converger una misma solución hacia el óptimo. No obstante, esta solución será mejorada necesariamente por secciones, y según el criterio que utilicemos para seleccionar el músculo a *entrenar* (mejorar) podremos añadir la diversificación necesaria para que la metaheurística no se estanque en óptimos locales. Esto es, con el *criterio de selección*.

**Criterio de selección**

Una vez establecida la representación particular de la optimización muscular, nos queda una pregunta más. ¿Cómo escogemos el músculo que se va a entrenar de la solución?

Aquí es donde veo relevante remarcar el carácter general y ajustable de la metaheurística. No es un único criterio el que tenemos a nuestro alcance sino innumerables criterios, tales como: el músculo que más aporte a la función objetivo, el músculo con peor/mejor fitness proporcional a su tamaño, un músculo aleatorio…

Para la implementación yo escogeré el criterio de “músculo aleatorio” con tal de aportar diversificación. Al limitar la convergencia de la solución a un segmento aleatorio del individuo, obligamos a que el algoritmo explore soluciones que de otra manera no habría explorado si hubiéramos tenido en cuenta la solución al completo,

pues habrían sido eclipsadas por aquellas mejoras globales superiores a las posibles en el segmento restringido.

No obstante, por muy prometedor que pueda sonar el criterio de músculo aleatorio, es importante mencionar que no tiene por qué ser el mejor. Dependerá del espacio de búsqueda del problema el que nos interese diversificar o explotar más la solución.

Lo que sí será universal para todo criterio de selección es que, una vez no sea posible mejorar la solución a partir de un músculo, habrá que seleccionar el siguiente músculo más apropiado (según el mismo criterio).

### Criterio de mejora

Una vez seleccionado el músculo que vamos a entrenar, ¿cómo lo mejoramos? ¿Qué entendemos por mejora?

Nuevamente, la generalidad de la metaheurística daría pie a numerosas opciones, desde aplicar conocimiento del problema para mejorar la solución con una búsqueda más informada, hasta la hibridación con otros algoritmos que converjan la solución al óptimo local que supone el segmento restringido del músculo.

En la implementación haré uso de una especie de búsqueda local que consistirá en probar valores aleatorios para cada elemento del músculo y asignar aquellos que mejoren el fitness global de la solución. Aleatorios necesariamente, no barajados, ya que vamos a trabajar con un rango de valores *real*.

Esto supondrá un problema, y es que si la solución es inmejorable para un músculo el algoritmo ciclará indefinidamente probando aleatorios y malgastando evaluaciones. Para solventarlo, definiremos una constante denominada número máximo de fallos que determinará el número de veces que permitiremos que el algoritmo genere aleatorios que no mejoren el fitness. Alcanzado ese número máximo de fallos, consideraremos que la solución es inmejorable desde ese músculo y seleccionaremos el siguiente más apropiado (en nuestro caso, uno aleatorio).

## Pseudocódigo

Ya puestos sobre la mesa las principales particularidades de la metaheurística, esquematizaré el algoritmo con pseudocódigo para hacer más intuitivo su desarrollo. Se tendrá un número máximo de evaluaciones como criterio de parada, y la división en músculos de tamaño aleatorio se hará con una función dividirMúsculos que dará como resultado un vector de enteros que almacenará el comienzo (en el vector solución) de cada músculo. Los criterios de selección y mejora se establecerán como criterios genéricos, y los elementos del músculo se recorrerán secuencialmente en círculo aprovechando la aleatoriedad que ya implica la división y selección del músculo.

**Entrada:** ini vector solución inicial, dim dimensión de la solución, max_fails número máximo de fallos, max_evals número máximo de evaluaciones, num_musculos número de músculos.

1. Inicializar vector de enteros: distribución
2. distribución = dividirMúsculos(dim, num_musculos)
3. mejor = ini
4. evaluaciones = 0
5. **while** evaluaciones < max_evals **do**
    I. índiceMúsculo, tamMúsculo = criterioSelección(distribución)
    II. fallos = 0
    III. **while** fallos < max_fails **do**
        1. aux = criterioMejora(mejor)
        2. **if** f(aux) es mejor que f(mejor) **do**
            I. mejor = aux
        3. **else do**
            I. fallos = fallos + 1
        4. **end**
        5. evaluaciones = evaluaciones + 1
        6. índiceMúsculo = (índiceMúsculo + 1) *mod* tamMúsculo
    IV. **end**
6. **end**
7. **return** mejor

Cabe mencionar que en la implementación no he definido número máximo de músculos porque siempre escojo un número aleatorio de ellos, pero sí es un parámetro a tener en cuenta en la metaheurística original.

# Parte II. Experimentación con la competición CEC2017.

## Breve descripción

CEC2017 es un benchmark para la comparación de algoritmos en problemas básicos de minimización de funciones, el cual usaremos para evaluar el rendimiento de la *optimización muscular*. Incluye 29 distintas funciones además de cuatro tipos de dimensiones (10, 30, 50, 100) del vector solución, con un rango de valores real [-100, 100] por cada elemento del vector. Dichas funciones vienen definidas con un óptimo previamente calculado, por lo que evaluaremos el rendimiento dada la distancia a ese óptimo (el error). Más concretamente, el error promedio tras diez ejecuciones.

## Resultados

Empezaré comparando con algoritmos clásicos de la competición como PSO (particle swarm optimization) y DE (differential evolution), ya que si la OM (optimización muscular) no es capaz de superar estos dos no tendría sentido compararla con otros algoritmos más complejos y eficientes. Estos son los resultados promedios para dimensión 10, 30 y 50 (respectivamente):

| | DE | Muscular | PSO |
|---|---|---|---|
| Best | 20 | 6 | 4 |
| F01 | 0 | 36817.39 | 52551101 |
| F02 | 0 | 11.3 | 1 |
| F03 | 0 | 14223.46 | 1988.879 |
| F04 | 0.000111 | 20.18636 | 46.84269 |
| F05 | 115.0819 | 16.22523 | 32.12004 |
| F06 | 34.59707 | 0.070555 | 10.01006 |
| F07 | 38.48056 | 27.97524 | 42.75206 |
| F08 | 29.8292 | 18.81044 | 22.03181 |
| F09 | 193.788 | 33.82614 | 56.86467 |
| F10 | 359.6808 | 701.0663 | 1076.895 |
| F11 | 0.019419 | 44.29752 | 38.42737 |
| F12 | 4.931086 | 2532812 | 2516941 |
| F13 | 5.988143 | 13545.67 | 8408.6 |
| F14 | 0.052398 | 5754.426 | 99.92584 |
| F15 | 0.060603 | 11106.71 | 2065.841 |
| F16 | 456.0592 | 196.1595 | 141.4558 |
| F17 | 23.50453 | 58.39495 | 64.9719 |
| F18 | 0.0363 | 10094.97 | 14840.1 |
| F19 | 0.005192 | 9810.679 | 3219.606 |
| F20 | 383.6905 | 5.517741 | 84.4391 |
| F21 | 188.8937 | 228.4282 | 131.952 |
| F22 | 100.481 | 538.0339 | 77.40418 |
| F23 | 809.7517 | 345.4782 | 330.4044 |
| F24 | 100 | 269.4298 | 181.0388 |
| F25 | 404.0113 | 414.5996 | 447.8069 |
| F26 | 270.5882 | 1168.457 | 372.9316 |
| F27 | 389.7283 | 430.6798 | 413.4232 |
| F28 | 351.7333 | 530.8436 | 469.7623 |
| F29 | 237.5455 | 382.2396 | 319.2936 |
| F30 | 80512.17 | 704015.9 | 635248.2 |

| | DE | muscular | PSO |
|---|---|---|---|
| Best | 19 | 9 | 2 |
| F01 | 49094.72 | 80654.71 | 4.18E+09 |
| F02 | 1.31E+19 | 5.8E+12 | 1 |
| F03 | 3481.079 | 187410.5 | 54534.28 |
| F04 | 84.30386 | 99.28618 | 1183.191 |
| F05 | 201.4981 | 113.489 | 217.0438 |
| F06 | 6.320261 | 0.058741 | 36.93872 |
| F07 | 233.4222 | 121.5427 | 359.5821 |
| F08 | 189.372 | 131.5961 | 174.5205 |
| F09 | 65.2973 | 1780.91 | 2841.846 |
| F10 | 3763.561 | 3042.803 | 6938.089 |
| F11 | 79.5828 | 5425.079 | 1207.135 |
| F12 | 325765.2 | 2316459 | 3.59E+08 |
| F13 | 153.6136 | 336901.1 | 45075854 |
| F14 | 71.00224 | 3827602 | 305870.5 |
| F15 | 62.55673 | 21251.33 | 273710.1 |
| F16 | 1319.233 | 1303.454 | 1568.291 |
| F17 | 480.8806 | 784.6265 | 472.5476 |
| F18 | 61.22245 | 4141452 | 2170364 |
| F19 | 35.72268 | 17871.89 | 1260194 |
| F20 | 275.1128 | 643.1905 | 462.1451 |
| F21 | 325.4941 | 311.2543 | 411.2916 |
| F22 | 100.2235 | 3324.924 | 1026.573 |
| F23 | 534.6092 | 464.9289 | 640.4298 |
| F24 | 605.9018 | 688.369 | 709.5056 |
| F25 | 387.028 | 390.9068 | 686.3826 |
| F26 | 403.7673 | 2162.527 | 3369.357 |
| F27 | 492.5198 | 541.2346 | 806.8056 |
| F28 | 394.2696 | 409.3278 | 1105.036 |
| F29 | 1025.095 | 903.3968 | 1409.126 |
| F30 | 3656.552 | 16839.47 | 13585755 |

| | DE | Muscular | PSO |
|---|---|---|---|
| Best | 12 | 17 | 1 |
| F01 | 2.04E+08 | 90910.04 | 1.82E+10 |
| F02 | 5.16E+52 | 1.69E+31 | 1 |
| F03 | 75969.91 | 387119.5 | 100772.2 |
| F04 | 232.2053 | 136.4737 | 4091.419 |
| F05 | 397.7698 | 233.5132 | 432.8081 |
| F06 | 11.80296 | 0.068203 | 54.5041 |
| F07 | 478.1017 | 228.2192 | 819.6689 |
| F08 | 406.9707 | 197.7843 | 407.7498 |
| F09 | 3755.83 | 3382.808 | 16198.75 |
| F10 | 11274.47 | 5242.141 | 12945.71 |
| F11 | 217.3029 | 11980.35 | 3151.373 |
| F12 | 1.38E+08 | 6356802 | 6.06E+09 |
| F13 | 27264.28 | 40196.94 | 5.26E+08 |
| F14 | 168.3185 | 5938001 | 1558397 |
| F15 | 508.9713 | 22522.38 | 5790072 |
| F16 | 3047.474 | 1944.92 | 2744.694 |
| F17 | 1827.133 | 1612.104 | 1651.838 |
| F18 | 24198.42 | 15406831 | 12034534 |
| F19 | 132.1995 | 13617.66 | 13358179 |
| F20 | 938.4169 | 1325.976 | 1282.219 |
| F21 | 617.6803 | 435.5598 | 665.5464 |
| F22 | 170.7379 | 5842.674 | 12582.31 |
| F23 | 843.4995 | 662.5361 | 1097.931 |
| F24 | 884.9783 | 1281.854 | 1206.235 |
| F25 | 566.7174 | 534.0106 | 3069.067 |
| F26 | 574.5069 | 3158.776 | 7890.316 |
| F27 | 576.5073 | 893.182 | 1743.583 |
| F28 | 501.3251 | 491.5227 | 3468.41 |
| F29 | 2217.738 | 1944.737 | 3565.966 |
| F30 | 2386399 | 1351845 | 3.37E+08 |

Para dimensión 10:

1. DE devuelve la mejor solución 20 veces.
2. OM devuelve la mejor solución 6 veces.
3. PSO devuelve la mejor solución 4 veces.

Para dimensión 30:

1. DE devuelve la mejor solución 19 veces.
2. OM devuelve la mejor solución 9 veces.
3. PSO devuelve la mejor solución 4 veces.

Para dimensión 50:

1. OM devuelve la mejor solución 17 veces.
2. DE devuelve la mejor solución 12 veces.
3. PSO devuelve la mejor solución una vez.

Por lo que podemos observar, la implementación de optimización muscular rinde mejor cuanto mayor es la dimensión sobre la que trabaja. Probemos entonces a enfrentar la OM con algoritmos más modernos para dimensión 50, concretamente AEO (artificial ecosystem-based optimization) y SSA (singular spectrum análisis). Estos son los resultados (10, 30, 50):

| | AEO | Muscular | SSA |
|---|---|---|---|
| Best | 1 | 24 | 6 |
| F01 | 5.23E+08 | 90910.04 | 5922.35 |
| F02 | 1 | 1.69E+31 | 1 |
| F03 | 96046.64 | 387119.5 | 0.000594 |
| F04 | 445.9256 | 136.4737 | 140.9477 |
| F05 | 423.5969 | 233.5132 | 448.4464 |
| F06 | 78.49611 | 0.068203 | 70.69212 |
| F07 | 1092.223 | 228.2192 | 946.685 |
| F08 | 445.3787 | 197.7843 | 441.1868 |
| F09 | 22018.66 | 3382.808 | 16171.41 |
| F10 | 10600.73 | 5242.141 | 8333.179 |
| F11 | 1148.874 | 11980.35 | 253.1243 |
| F12 | 4.14E+08 | 6356802 | 10994175 |
| F13 | 9242019 | 40196.94 | 176291.6 |
| F14 | 462824.5 | 5938001 | 26742.62 |
| F15 | 495592.7 | 22522.38 | 74959.57 |
| F16 | 3792.471 | 1944.92 | 2703.787 |
| F17 | 2073.382 | 1612.104 | 2188.364 |
| F18 | 2990373 | 15406831 | 212911.7 |
| F19 | 5298829 | 13617.66 | 400903 |
| F20 | 1549.698 | 1325.976 | 1610.491 |
| F21 | 781.8707 | 435.5598 | 747.7839 |
| F22 | 11236.07 | 5842.674 | 9165.073 |
| F23 | 1546.376 | 662.5361 | 1464.919 |
| F24 | 1668.612 | 1281.854 | 1563.594 |
| F25 | 822.9434 | 534.0106 | 559.1655 |
| F26 | 8047.064 | 3158.776 | 10827.68 |
| F27 | 1792.022 | 893.182 | 1315.574 |
| F28 | 843.0862 | 491.5227 | 823.6175 |
| F29 | 4769.452 | 1944.737 | 4358.074 |
| F30 | 2E+08 | 1351845 | 20590877 |

1. OM devuelve la mejor solución 24 veces.
2. SSA devuelve la mejor 6 veces.
3. AEO devuelve la mejor una vez.

La optimización muscular supera con aun más ímpetu los nuevos dos algoritmos con los que se compara, dando la mejor solución para la mayoría de funciones en dimensión 50. Lo último que quedaría por comprobar sería, por tanto, su rendimiento en contraste con algoritmos más competitivos. Concretamente, se comparará con jSO y EBOwithCMAR. Estos son los resultados:

| | EBOwithCMAR | Muscular | jSO |
|---|---|---|---|
| **Best** | 17 | 0 | 16 |
| **F01** | 0 | 90910.04 | 0 |
| **F02** | 1 | 1.69E+31 | 0 |
| **F03** | 0 | 387119.5 | 0 |
| **F04** | 42.86363875 | 136.4737 | 56.21258 |
| **F05** | 7.584556986 | 233.5132 | 16.40531 |
| **F06** | 8.54E-08 | 0.068203 | 1.09E-06 |
| **F07** | 57.88431782 | 228.2192 | 66.49652 |
| **F08** | 7.911394759 | 197.7843 | 16.96232 |
| **F09** | 0 | 3382.808 | 0 |
| **F10** | 3114.736471 | 5242.141 | 3139.758 |
| **F11** | 26.36143449 | 11980.35 | 27.93855 |
| **F12** | 1938.754196 | 6356802 | 1680.563 |
| **F13** | 41.40139101 | 40196.94 | 30.5989 |
| **F14** | 31.21489729 | 5938001 | 24.96366 |
| **F15** | 29.35824894 | 22522.38 | 23.86433 |
| **F16** | 346.3637327 | 1944.92 | 450.5211 |
| **F17** | 274.7809067 | 1612.104 | 282.8672 |
| **F18** | 32.03371147 | 15406831 | 24.28282 |
| **F19** | 24.47839876 | 13617.66 | 14.13861 |
| **F20** | 147.2183591 | 1325.976 | 140.1016 |
| **F21** | 210.6179363 | 435.5598 | 219.1995 |
| **F22** | 365.3677327 | 5842.674 | 1487.236 |
| **F23** | 434.0470247 | 662.5361 | 430.0837 |
| **F24** | 506.4619198 | 1281.854 | 507.45 |
| **F25** | 488.6050682 | 534.0106 | 480.878 |
| **F26** | 705.5736088 | 3158.776 | 1128.779 |
| **F27** | 522.3751004 | 893.182 | 511.2716 |
| **F28** | 466.5108345 | 491.5227 | 459.8068 |
| **F29** | 347.3438812 | 1944.737 | 362.9353 |
| **F30** | 618165.3976 | 1351845 | 601051.7 |

1. EBOwithCMAR devuelve la mejor solución 17 veces.
2. jSO devuelve la mejor solución 16 veces.
3. OM nunca devuelve la mejor solución.

**Comentario**

Para empezar, es destacable mencionar el comportamiento del algoritmo respecto a la dimensión del vector solución. Conforme mayor era esa dimensión, más capaz era de encontrar el óptimo, y considero que es algo que podemos achacar a la aleatoriedad de los criterios de selección y mejora que he escogido para la implementación.

El seleccionar y mejorar los músculos aleatoriamente implica que el algoritmo dependa en cierta medida del azar para escoger caminos óptimos. Esto se hace más evidente si vemos los resultados para cada ejecución. Por ejemplo, para la función 2 (Shifted and Rotated Zakharov Function), el algoritmo usualmente daba con el óptimo (error 0), pero en ciertas ejecuciones se dieron picos de hasta 100 de error.

No obstante, tiene sentido que esa dependencia del azar se vea reducida conforme aumente la dimensión del vector solución, ya que se tienen en cuenta más valores aleatorios y por tanto hay más probabilidad de que el vector dé con el camino correcto al óptimo.

Es coherente entonces que sea capaz de superar algoritmos como DE, PSO, SSA y AEO. En altas dimensiones la aleatoriedad se pone a su favor para lograr un equilibrio diversificación/explotación suficientemente potente para alcanzar óptimos decentes.

No obstante, no es perfecto. En cuanto es enfrentado a algoritmos profesionales como EBOwithCMAR y jSO no es capaz de obtener ni una sola solución mejor que la de cualquiera de los dos. Es decir, siempre está por debajo que los dos algoritmos a la vez.

Es difícil discernir a qué se puede deber esta inferioridad tan considerable, más allá de la diferencia de mi experiencia y conocimiento con la de los autores de EBOwithCMAR y jSO. Son numerosos los factores a tener en cuenta para justificar esos rendimientos (forma de cada función, funcionamiento de los algoritmos, parámetros…), sin embargo, considero que hay unos hechos generales que seguramente sean la causa (o al menos en parte) de esa inferioridad.

Para empezar, los criterios escogidos. Por falta de tiempo no se ha experimentado con criterios de selección que, aunque más complejos, podrían haber sido más prometedores a la hora de guiar la búsqueda, ni tampoco se ha probado a mejorar con alternativas que llevase a converger la solución más deprisa desde el músculo. Lógicamente, estas opciones no tendrían porqué haber supuesto un mejor rendimiento, pero sí que nos habría dado una idea más global de qué es efectivo y qué no para la metaheurística.

Además, hay que tener en cuenta que la optimización muscular es un algoritmo bastante sencillo. De la misma manera que búsqueda local es un algoritmo simple que se vuelve interesante al ser hibridado con algoritmos como los genéticos, la optimización muscular podría tener más potencial si se hibridase con un algoritmo que aprovechase sus puntos fuertes y compensase sus carencias.

# Parte III. Propuesta de mejora con hibridación.

## Tipos de hibridación y opciones

Habiendo establecido las posibles causas del subóptimo rendimiento de la optimización muscular, comenzamos comentando la mejora con hibridación.

Escoger el algoritmo a hibridar ya es tarea ardua porque supone valorar lo que nos aporta las múltiples metaheurísticas que vayamos a tener en cuenta y contraponerlas a las características de la optimización muscular. No obstante, hay una dificultad añadida y es que hay dos enfoques principales para la hibridación.

El primero es el planteamiento clásico de utilizar optimización muscular como algoritmo secundario dentro de otro. Sabiendo que OM es una metaheurística principalmente de explotación, podría ser interesante introducirla en algoritmos principalmente de diversificación como los genéticos. La manera de hacerlo, de nuevo, tiene diversos enfoques.

Se podría entrenar muscularmente a un número pequeño o aleatorio de la población cada cierto número de generaciones, o al mejor de la población, o a todos para un solo músculo aleatorio… Las posibilidades son innumerables y la que escojamos dependerá del problema que pretendamos resolver.

El segundo enfoque es el del criterio de mejora. Sabemos que, una vez seleccionado el músculo sobre el que vamos a mejorar la solución, el algoritmo debe seguir un criterio de mejora para *entrenar* ese músculo. Como normalmente es en la división y selección de músculo donde se aporta la diversificación al algoritmo, lo interesante en este punto sería utilizar técnicas de explotación como búsqueda local que converjan la solución todo lo posible dentro del segmento restringido del músculo. Es decir, hibridar con algoritmos de explotación de tal manera que hagan converger la solución pero solo modificando elementos del segmento permitido que es el músculo.

## Hibridación con algoritmo genético generacional

Finalmente, he optado por hibridar la optimización muscular con un algoritmo genético generacional no elitista bajo el primer enfoque.

Si estuviésemos tratando un problema concreto, me habría informado sobre el espacio de búsqueda y habría planteado una serie de alternativas a probar que considerase prometedoras. Sin embargo, como estamos lidiando con un problema genérico con todo tipo de funciones, no me quedó otra que optar por la hibridación que mejor creía que iba a funcionar a nivel general, basándome nuevamente en el fenómeno de la naturaleza de la que se originó la metaheurística en un primer momento.

El razonamiento es el siguiente: volviendo al hombre prehistórico, sabemos que él desarrollará las aptitudes que le sean necesarias para realizar eficientemente la tarea que es sobrevivir. No obstante, él forma parte de un todo más grande que el simple hecho de la supervivencia. Su capacidad para mantenerse con vida le hará un individuo más apto para cumplir con la gran segunda tarea de toda especie: reproducirse. Así, si consigue sobrevivir y destacar entre el resto de individuos, se asegurará que sus genes se extiendan durante generaciones e influenciará las poblaciones del futuro.

Es por esto que una hibridación con un algoritmo genético generacional es la opción más interesante a primera vista, porque nos permite reflejar con aún más autenticidad el fenómeno de la supervivencia y reproducción del más apto que se ha estado dado desde la existencia de los seres vivos.

La manera más adecuada que consideré para la implementación de esta hibridación era la de entrenar por optimización muscular a todos los individuos de cada generación, pero para un solo músculo escogido aleatoriamente (misma distribución entre individuos, distinto músculo seleccionado). De esta manera se reflejaría cómo cada individuo de la población desarrolla una aptitud distinta para sobrevivir.

No obstante, sabemos que en la naturaleza el entrenamiento no es un factor heredable para los descendientes, y aunque podamos ignorar esa particularidad en la implementación se podría dar una convergencia prematura de la población (por estar explotando todos los individuos desde el primer momento y en cada generación). Por lo tanto, se resolverá de la siguiente manera: se entrenará un músculo aleatorio de cada individuo de la población pero no se almacenará al individuo entrenado, solo el valor de fitness tras su entrenamiento y la mejor solución entrenada. Es decir, mantendremos las soluciones como estaban previas a la optimización muscular, pero las compararemos dado su fitness de entrenamiento, de esta manera escogiendo las

soluciones más “musculadas” para que se reproduzcan pero impidiendo que dicho entrenamiento lo hereden los hijos (como sería en la naturaleza).

Y manteniendo esa fidelidad con la naturaleza, el algoritmo no será elitista. Es decir, se guardará la mejor solución encontrada hasta el momento (la versión entrenada) pero no será reincluida en la población en ningún momento.

## Pseudocódigo

Establecido el razonamiento detrás de la hibridación con los algoritmos genéticos generacionales, procedo a incluir pseudocódigo para concretar la implementación. Los operadores de selección, cruce y mutación particulares de los algoritmos genéticos se plantearán como operadores genéricos para simplificar el ejemplo. La función de optimizaciónMuscular se entenderá como una modificación del algoritmo definido previamente, donde solo se mejora un músculo por individuo; en cuanto no es posible mejorar más la solución a partir de ese músculo, el algoritmo termina.

**Entrada:** num_c número de cromosomas, dim dimensión de la solución, max_evals número máximo de evaluaciones, num_musculos número de músculos para OM, max_fails número máximo de fallos para OM.

1. Inicializar matriz vacía de soluciones P
2. **while** tamaño de P < num_c **do**
   I. S = generarSoluciónAleatoria(dim)
   II. añadir S a P
3. **end**
4. dis = dividirMúsculos(dim, num_musculos)
5. Evaluaciones, mejorS = evaluar(optimizaciónMuscular(P, dis, max_fails))
6. n_evaluaciones = num_c
7. **while** n_evaluaciones < max_evals **do**
   I. Padres = seleccionar(P, Evaluaciones)
   II. Intermedia = cruzar(Padres)
   III. Hijos = mutar(Intermedia)
   IV. P = Hijos
   V. Evaluaciones, mejorActual = evaluar(optimizaciónMuscular(P, dis, max_fails))
   VI. n_evaluaciones = n_evaluaciones + num_c
   VII. **if** f(mejorActual) mejor que f(mejorS) **do**
      1. mejorS = mejorActual
   VIII. **end**
8. **end**
9. **return** mejorS

# Parte III. Propuesta de mejora sobre el diseño.

## El problema

Dejando de lado la posibilidad de hibridar (pues se ha comentado en el apartado anterior) y el hecho de que toda mejora con criterio de una metaheurística requiere, por lo menos, un esfuerzo de análisis teórica y empíricamente, hay un problema principal respecto a la optimización muscular.

Ese problema es su generalidad. Tenemos una metaheurística cuya principal función es dividir una solución y resolver segmento a segmento. Es una definición suficientemente genérica como para que no se pueda plantear una mejora intrínsecamente buena, y que en caso de que se plantease no sería realmente una mejora sino transformarla en una metaheurística completamente distinta.

¿Qué nos queda entonces? Trabajar con las partes más concretas del algoritmo. Es decir, con *cómo* dividimos la solución y *cómo* resolvemos. No será tanto una mejora sobre el diseño, sino una selección con más criterio de los operadores de la metaheurística.

## División de la solución

Hasta ahora habíamos planteado la división como simplemente partir la solución en $n$ segmentos de tamaño aleatorio. Es una manera efectiva, y similar al fenómeno natural de donde se inspira la metaheurística (los gemelos son más grandes que los bíceps, pero los gemelos son iguales entre sí).

No obstante, puede ser problemático. Podrían darse ciertas distribuciones donde un solo músculo contiene la mayor parte de los elementos de la solución, o músculos demasiado insignificantes que nada más que incluyen un solo elemento. Sería entonces de nuestro interés limitar esa aleatoriedad, por ejemplo, haciendo que los músculos no pudiesen ser más grandes que la mitad de la solución al completo, o que tuviesen que incluir por lo menos dos elementos.

Esto ya nos aporta opciones por la parte de la aleatoriedad, pero no tenemos porqué limitarnos a ello. En algunos problemas quizás nos podría interesar dividir la solución en músculos de tamaños fijos e iguales, o incluso podríamos utilizar información del propio problema para saber qué elementos del vector solución son más relevantes para la función objetivo y organizarlos en músculos según esa relevancia.

Tenemos multitud de opciones interesantes, y no es trivial escoger ya que la división de la solución es la base del algoritmo. Yo escogí una simple división en músculos de

tamaño aleatorio por la generalidad del problema que estábamos tratando, pero sería conveniente valorar otras opciones en caso de estar lidiando con un problema real.

## Criterios

Volvemos entonces a mencionar brevemente las posibilidades de los criterios de selección y mejora, ya que, al igual que la división de los músculos, son clave para adecuar el algoritmo al problema.

El criterio de selección es el principal diversificador del algoritmo. Si queremos una alta exploración nos interesará, por ejemplo, seleccionar aleatoriamente, o escoger aquel músculo menos relevante para el fitness de la solución (pues así tarda más en converger). Si queremos que el algoritmo explore mínimamente, nos interesará seleccionar aquel que ya tenga un fitness alto proporcional a su tamaño, o que sea el más relevante para el fitness global de la solución (tarda menos en converger). Nuevamente, dependerá del espacio de búsqueda que escojamos un criterio de selección u otro.

No obstante, ya una vez hemos seleccionado músculo y nos hallamos en el criterio de mejora, no tiene sentido obstaculizar la explotación con tal de diversificar. La diversificación deseada ya se debe haber aportado en la selección, por lo que ahora nos interesa entrenar ese músculo lo más eficientemente posible para no malgastar evaluaciones (y tiempo de ejecución). Esto se puede lograr con algoritmos de rápida convergencia como búsqueda local, que exploten la solución hasta que no se pueda mejorar más en el segmento restringido del músculo.

## Parte Extra. Compilación y ejecución del código.

La metaheurística fue implementada en un sistema Linux usando como base el código de Daniel Molina de la competición CEC2017 ( https://github.com/dmolina/cec2017real/ ).

Para ejecutar mi implementación, será necesario tener instalado un sistema Linux, además de las herramientas CMake y Make, y seguir los siguientes pasos:

1. Abrir una consola en el directorio *code* que se encuentra en el zip de la entrega.
2. Ejecutar el comando “cmake .” que preparará el makefile.
3. Ejecutar el comando “make” que compilará el código.
4. Darle permisos al ejecutable muscular_ggarredondo con el comando “chmod +x muscular_ggarredondo”
5. Ejecutar con el comando “./muscular_ggarredondo”

Hecho esto se empezarán a mostrar los resultados del algoritmo para las 29 funciones, las dimensiones 10, 30 y 50, y las 10 semillas, los cuales se almacenarán en la carpeta “results_muscular” como un archivo de texto txt.

Si se tuviese interés en obtener el Excel con el cual se comparó con el resto de algoritmos de la competición, sería tan sencillo como ejecutar el comando “python extract.py results_muscular” (es necesario tener instalado el intérprete de Python), el cual generará un Excel por cada tipo de dimensión en el directorio *results_muscular*.