Aplicación de algoritmos genéticos a problemas de optimización

UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo de fin de grado
2018 - 2019
Grado en matemáticas

Septiembre 2019

Dionisio Martínez Alberquilla

Director:
Fernando Rubio Diez

Resumen

Las técnicas evolutivas en computación han avanzado mucho en los últimos años dado que han demostrado su eficiencia en la resolución de problemas de alto coste computacional (exponencial en muchos casos). Los algoritmos genéticos, como parte de la computación evolutiva, fueron desarrollados, en primer lugar, por Henry Holland a finales de los años 70. Dada su particular generalidad, estas técnicas pueden aplicarse a una gran cantidad de problemas reales tales como el *marketing*, diseño de productos y problemas de optimización en general.

La mayoría de los problemas de optimización de la vida real poseen un alto grado de complejidad computacional, es decir, forman parte de la clase $\mathbf{NP} - \mathsf{hard}$. Es por ello, que los investigadores traten de encontrar mejores algoritmos. El problema surge cuando dichos algoritmos no se encuentran, o incluso peor, cuando ni siquiera existen. Es aquí cuando los algoritmos genéticos muestran su potencial, permitiéndonos aproximar soluciones de los problemas, o incluso encontrar el óptimo, en una cantidad razonable de tiempo.

El comportamiento de estos algoritmos trata de imitar la evolución natural; comienzan con una población aleatoria (posibles soluciones) y posteriormente, los operadores naturales (cruce, mutación) son aplicados a cada individuo de la población, reemplazando los antiguos miembros con la esperanza de que, con el tiempo, los nuevos resulten mejor adaptados al medio (restricciones), dando lugar a soluciones válidas.

Como puede observarse, no existe un criterio de parada estándar, simplemente se escoge el más adecuado para cada situación. En función de cada una, dicho criterio puede ser un número fijo de iteraciones, una cota superior del nivel de adaptación, etc.

A lo largo de este trabajo expondremos los conceptos básicos de la programación genética, desde los operadores fundamentales que involucran aleatoriedad, hasta otros más avanzados como el algoritmo *qNNR* cuando se considere la formación de nichos poblacionales. La principal idea de este trabajo es cuantificar la eficiencia de estos algoritmos aplicados a problemas reales. Para ello, nos centraremos en un problema en particular, aunque general en su aplicación, conocido como *Advertisement Placement Optimization Problem*.

Palabras clave: Algoritmo genético, optimización, *marketing*, diversidad genética, operador genético.

Abstract

The evolutionary techniques in computation have grown in the past years since they have shown their efficiency in high-cost computational tasks (exponential complexity in most of the cases). Genetic algorithms, as part of evolutionary computation, have been first developed by Henry Holland in the late 70s. Due to its particularly generality, these techniques can be applied to a wide range of real-life problems, such as marketing, product design and optimization problems in general.

Most of the real-life problems have a huge computational complexity, *i.e.*, they belong to $\mathbf{NP} - \mathsf{hard}$. That is why a lot of researchers are trying to find better algorithms. The problem arises when none of these algorithms are found, or even worse when they don't even exist. Here is where genetic algorithms are most useful, allowing us to find approximate solutions for the problems, or maybe the optimum, in a reasonable time.

The behavior of such algorithms mimics the natural selection: they start with a random population of individuals (candidates to solution), then some natural operators (such as crossover, mutation) are applied to the population, replacing the old members hoping the new ones become more adapted to the medium (restrictions) with enough time, resulting in a valid solution.

As can be seen, there are no obvious stop criteria for these algorithms; one simply chooses the more suitable one. Depending on conditions, these criteria can be a fixed number of iterations, an upper bound for fitness...

Throughout this work we will expose the basic concepts of genetic programming: from the basic operators that involve randomness to more advanced ones, like q-nearest neighbor algorithms when considering niching in population. The main idea of this work is trying to measure the efficiency of these algorithms when applied to real-life problems. For this purpose we have chosen a very general one from marketing, usually called *Advertisement Placement Optimization Problem.*

Keywords: Genetic algorithm, optimization, marketing, genetic diversity, genetic operator.

Índice

1. Introducción

Desde hace años es bien sabido que una gran cantidad (por no decir la gran mayoría) de los problemas de optimización son *difíciles* de resolver, es decir, los algoritmos para hallar una solución exacta pertenecen a la clase $\mathbf{NP}-\mathsf{hard}$. Dado que no podemos calcular en tiempo polinómico soluciones a dichos problemas, el esfuerzo se centra en la búsqueda de algoritmos que las aproximen, dando lugar al estudio de las clases de aproximación. Los problemas que trataremos en este trabajo pertenecen a la clase $\mathsf{Log}-\mathsf{APX}$: un problema pertenece a esta clase si su ratio de aproximación (cociente entre solución exacta y aproximada) puede acotarse por una función logarítmica en el tamaño del problema. La programación evolutiva es ampliamente aplicada a problemas $\mathbf{NP}-\mathsf{hard}$, ya que permite realizar una búsqueda *guiada* en el espacio de soluciones, reduciendo drásticamente la complejidad de una búsqueda exhaustiva y alcanzando soluciones que aproximan (o alcanzan) a las soluciones óptimas en un tiempo razonable.

Como principal característica, las técnicas evolutivas se basan en imitar el mecanismo de selección natural, aplicándolo a problemas concretos. Un algoritmo genético, como constituyente de estas técnicas, trata de aplicar el mecanismo de selección natural, partiendo de una población de individuos (posibles soluciones) a la cual se hace evolucionar con operadores propios de la biología (mutación, cruce, etc); todo esto sometido a una selección en forma de una función (conocida como *fitness*) que evalúa cuan bueno es cada individuo. Así, con el paso del tiempo las soluciones mejoran, aproximándose a la solución óptima.

Debido a la naturaleza de estos algoritmos, la aleatoriedad forma un papel fundamental en todo el proceso. Al ser algoritmos que aproximan soluciones, no existe un criterio de parada definido, principalmente porque no se dispone de mecanismos para determinar si la solución actual es óptima. Es por ello que pueden darse varios criterios, siendo el más usado fijar un número de iteraciones máximas, puesto que *a priori* no suele conocerse una cota de la solución; pero si este fuese el caso, puede tomarse como criterio que el mejor valor *fitness* de la población sobrepase cierto umbral. Además, al ser algoritmos que mejoran la solución en cada iteración, tomar un número fijo de iteraciones permite adaptar el tiempo de cómputo a cada situación. Por otra parte, estos algoritmos trabajan normalmente a nivel binario (representación) por lo que una gran cantidad de problemas pueden aproximarse con el uso de estos algoritmos sin prácticamente modificación alguna en su funcionamiento. Existen otras variantes como la representación real o en forma de árbol. Esta última permite aplicar estos algoritmos a problemas que no tienen una representación directa en forma de dígitos. En este trabajo hemos tomado la representación binaria ya que es la más natural que presenta el problema a resolver, pero cabe destacar que existen numerosos estudios sobre cómo afecta la representación a la funcionalidad de los algoritmos genéticos.

Los problemas que estudiamos en detalle provienen del ámbito del *marketing*, pero su estructura es lo suficientemente general como para extenderse a numerosos ámbitos en los que se desee localizar un reparto óptimo de bienes, atendiendo a ciertas restricciones y optimizando los recursos disponibles.

2. Objetivos y metodología

El principal objetivo de este trabajo es medir la efectividad de los algoritmos genéticos aplicados al campo de la optimización con restricciones. En el caso de estudio, aplicamos un algoritmo genético a los problemas APO (*Advertisement Placement Optimization*) y PAPO (*Probabilistic Advertisement Placement Optimization*) que describiremos más adelante. El marco de referencia del que disponemos para comparar nuestros resultados viene dado en [RRR16] donde se exponen los resultados obtenidos por un algoritmo genético en comparación con técnicas heurísticas. Puesto que no disponemos de las mismas instancias de los problemas, nos basaremos en las mencionadas heurísticas para las comparaciones, obteniendo así resultados consistentes dado el determinismo que presentan las heurísticas en comparación con el no determinismo de los algoritmos genéticos.

Con respecto a la metodología usada, nos basaremos en una implementación propia del algoritmo genético en el lenguaje C++, sin uso de librerías externas. Las pruebas se han realizado en dos plataformas diferentes, bajo GNU/Linux[1] y bajo Apple[2]. Se han implementado diversos operadores tanto a nivel de ADN como de población que nos han permitido hacer un estudio para determinar cuál (o cuáles) de ellos es el más apropiado para estos problemas en concreto. El código fuente de la implementación, así como las instancias de los problemas y los datos obtenidos pueden consultarse en la siguiente dirección: github.com/dionimar/genetic-algorithm-marketing.

El desarrollo de este trabajo comienza con una exposición de los algoritmos genéticos en su forma más general, detallando todos los pasos del algoritmo. Más tarde, se expondrán los problemas específicos y, posteriormente, analizaremos su eficacia y algunos aspectos destacados de cada operador.

[1] Debian 9, GNU/Linux, Intel core i5-4440
[2] Apple/OSX, Intel core i5-7360U

3. Algoritmo genético general

En esta sección describiremos el funcionamiento de un algoritmo genético en su versión general, es decir, daremos el esquema en el cual se basan los algoritmos genéticos.

Estos algoritmos forman parte de una rama de la computación, denominada computación evolutiva. Como su nombre indica, tratan de simular los mecanismos propios de la naturaleza con el fin de encontrar solución a ciertos problemas (típicamente de optimización). En este caso de estudio trabajaremos con la evolución, pero existen otros mecanismos naturales como las colonias de abejas o caminos de hormigas.

Con el fin de encontrar la solución a un problema aplicando estos algoritmos, la primera tarea será codificar los datos en términos sobre los que opera la propia evolución, es decir, ADN, individuos, población, etc.

Como sucede de forma natural, un mismo conjunto básico de símbolos (a nivel biológico ACTG) que podemos tomar como $\{0, 1\}$, puede dar lugar a diferentes especies, es decir, puede manifestarse de maneras muy distintas. La manera de tratar con esta dualidad es mediante el genotipo y fenotipo: cómo representamos la información, y qué información estamos representando. Por ejemplo, el vector $(1, 1, 1, 1, 0, 1, 1)$ puede representar al número 123 en binario, o puede hacer referencia a un vértice de un hipercubo en dimensión 7. Es por esto, que los algoritmos genéticos gozan de mucha generalidad y, por tanto, pueden aplicarse a problemas muy diferentes sin realizar prácticamente modificaciones.

Cuando los individuos de una especie están sometidos a un medio, la selección natural favorece la existencia de los individuos más adaptados (los que presenten una característica que los permita adaptarse mejor y tener mayor probabilidad de sobrevivir, por tanto, mayor probabilidad de transmitir esa información a futuras generaciones). Con el fin de imitar este proceso, haremos uso de una función (*fitness*) del espacio de soluciones en $\mathbb{R}$ que medirá el nivel de adaptación de cada individuo (cuan buena es la solución). Un mismo algoritmo genético resolverá problemas diferentes dependiendo de la función *fitness* que consideremos.

La información que contiene el ADN de un individuo en la naturaleza es parcialmente transferida a la siguiente generación por medios reproductivos, ya sean sexuales o asexuales. Dado que hacemos uso de este mecanismo para hacer funcionar estos algoritmos, podemos permitirnos estudiar el comportamiento de métodos no naturales de reproducción, por ejemplo, la obtención de un individuo a partir de tres antecesores (o más).

El nivel perfecto de adaptación no está determinado (suponiendo su existencia en sentido biológico), por lo que, como ya se ha mencionado, los criterios de parada en estos algoritmos suelen ser un número fijo de iteraciones, o bien, si se conoce un valor aproximado que puede tomar la solución, puede tomarse un valor fijo que ha de alcanzar la función *fitness*. La elección particular dependerá del contexto sobre el que apliquemos estas técnicas.

Así, el esquema general de un algoritmo genético se describe en el algoritmo 1 (cuyo diagrama puede verse en la figura 1); consta de varias partes diferenciadas, las cuales pueden resumirse en:

- Inicialización: habitualmente se parte de una selección aleatoria de individuos a modo de población inicial (población cero). En ella podemos introducir individuos con características conocidas *a priori*, pudiendo así adaptar mejor el algoritmo al problema concreto.
- Cómputo de la aptitud/adaptación (*evaluate*): para cada individuo en la población actual, calcularemos su nivel de adaptación al medio al que lo hemos expuesto mediante el uso de la función *fitness*.
- En cuanto al orden anterior, producto de los valores *fitness*, aplicaremos algún criterio de selección a los individuos para, posteriormente y mediante un operador a nivel genético, generar nuevos individuos que deseablemente se adapten mejor al medio que los correspondientes en la generación anterior. Los procesos que resumimos a continuación quedan englobados en la función *evolve*:
  - Selección: puesto que buscamos mejorar los individuos con el paso de generaciones, trataremos de seleccionar los mejores de cada generación para que produzcan nuevos descendientes, de modo que con el tiempo se transmitan las mejores características genéticas.
  - Cruce: una vez seleccionados dos individuos para su reproducción, aplicaremos los operadores de cruce para obtener uno nuevo. La función de estos operadores es combinar la información genética de los predecesores para, posteriormente, incluirla en el nuevo individuo. En este trabajo consideraremos únicamente reproducción sexual con dos elementos, aunque existen varias formas de reproducción (3 individuos, generación de dos o más hijos a partir de un número de individuos, reproducción asexual... [SD07])
  - Mutación: obligaremos a los nuevos individuos a que sufran una mutación (con una probabilidad determinada, en nuestro caso 0.1), imitando el proceso natural que además aportará diversidad genética a la población.
  - Remplazo: tras la creación de un nuevo individuo, debemos tener algún criterio que determine cómo posicionarlo en la nueva generación, por ejemplo, si se sustituye el peor o si se hace aleatorio.

Algoritmo 1: Genetic Algorithm

```
t ← 0
initialize(population(t))
evaluate(population(t))
while ¬stop-criteria do
    population(t+1) ← evolve(population(t))
    evaluate(population(t+1))
    t ← t + 1
end
```

A continuación, pasaremos a describir los principales operadores usados en los algoritmos genéticos y expondremos detalladamente el funcionamiento de cada uno.

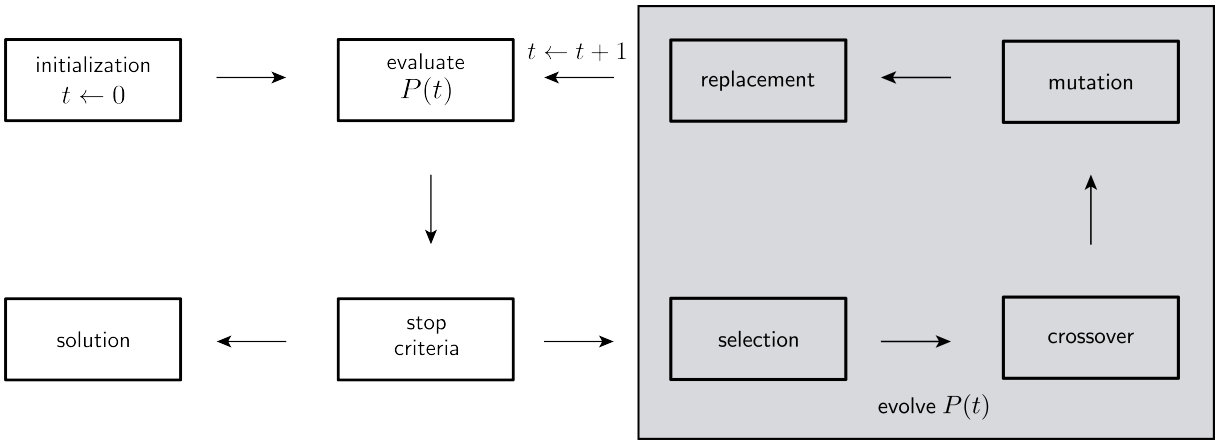

Figura 1: Diagrama algoritmo genético

3.1. Selección

El mecanismo de selección es de vital importancia en el rendimiento de un algoritmo genético. Su principal función es escoger los individuos a partir de los cuales obtendremos uno nuevo que formará parte de la siguiente generación. Este mecanismo tiene como propósito obtener en cada nueva generación individuos más aptos. Es necesario buscar algún criterio que permita que los nuevos individuos mejoren en las siguientes generaciones, pero que mantenga también la diversidad, de modo que no escoja únicamente los mejores para la reproducción. A esto se lo conoce como presión selectiva: a mayor presión el algoritmo mejorará considerablemente los nuevos descendientes en generaciones tempranas, pero provocará una convergencia prematura, es decir, que alcancemos una solución sub-óptima y una población con baja diversidad [Shi12]. Por el contrario, una baja presión provocará que el algoritmo emplee un tiempo innecesario en la búsqueda de soluciones aceptables al problema. Como medida de la diversidad en una población y basándonos en [GH07], tomaremos la media de las distancias de *Hamming* entre cada par de miembros m_i, m_j de la población:

$$\mathcal{H}(P) = \frac{2}{N * (N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \rho_H(m_i, m_j)$$

siendo

$$\rho_H(m_i, m_j) = \sum_{k=1}^{l} |m_{i,k} - m_{j,k}|$$

$$m_i, m_j \in \{0,1\}^l$$

Considerando los miembros de la población de longitud l, una situación ideal sería que la media de las diferencias entre todos los miembros de la población, $\mathcal{H}(P)$, estuviese cercana a l, lo cual indicaría una gran variedad genética ya que la función ρ_H mide la cantidad de símbolos diferentes entre los individuos m_i y m_j.

Existen diversas métricas para determinar la diversidad de una población, por ejemplo, la entropía de *Shannon* (véase [GH07] para una exposición más detallada sobre estas medidas), pero en este trabajo hemos optado por la expuesta anteriormente, pues su interpretación y análisis resultan más naturales.

3.1.1. Selección aleatoria

El mecanismo de selección aleatoria es simple: basta con tomar dos miembros de la población de forma aleatoria, siguiendo una distribución uniforme en el tamaño de la población, $U(0, M)$. Este procedimiento, si bien tiene la capacidad de distribuir uniformemente los candidatos para la reproducción por toda la población (siempre y cuando el tamaño de la población y el número de iteraciones lo permitan), no tiene en cuenta cuáles de estos están mejor adaptados.

3.1.2. Selección proporcional

La selección proporcional tiene como principal característica que considera el valor *fitness* de cada individuo para hacer una selección ponderada. El algoritmo calcula la suma de todos los valores *fitness* sobre la población, de modo que a cada individuo le asigna el cociente f_i/F, es decir, el valor proporcional que aporta su valor *fitness* al total de la población. Se crea una estructura adicional, denominada *pool*, donde añadiremos tantas repeticiones de un individuo como indique su contribución. Así, tomando una selección aleatoria y uniforme dentro de la *pool*, los individuos con mayor contribución tendrán más probabilidad de ser escogidos para la reproducción (sin eliminar a los de menor aportación, que tendrán una probabilidad mucho menor). Cabe destacar la posibilidad de elegir el tamaño de la *pool*, a mayor tamaño mayor precisión en la selección, pero también mayor tiempo en ejecución y consumo de memoria.

El algoritmo es el siguiente:

Algoritmo 2: Proportional Selection

```
population = {m_i | i ≤ N}
F ← Σ_{i=1}^{N} fitness(m_i)
foreach m_i ∈ population do
    times ← fitness(m_i) / F
    add(pool, m_i, times)
end
m_k, m_l ← random-selection(pool)
return m_k, m_l
```

Dado que la cantidad de apariciones de un individuo en la *pool* es proporcional a su contribución *fitness* respecto al total, si un individuo representa un porcentaje muy alto, la *pool* estará formada casi en su totalidad por dicho individuo, lo cual impide que el resto tenga opciones de selección. Esto es un gran inconveniente sobre todo en generaciones tempranas, ya que limitar tan bruscamente la reproducción produce una notable pérdida en la diversidad genética y, por consiguiente, una disminución de la capacidad de evolución del algoritmo. Además, hay una probabilidad muy alta de que hagamos el cruce del individuo mayoritario consigo mismo, lo cual provoca en operadores sexuales que la única diferencia entre el nuevo individuo y su progenitor sea una mutación.

3.1.3. Selección por ruleta

El mecanismo de selección por ruleta es muy similar al anterior; partimos de la misma idea de asignar a cada individuo una probabilidad de selección proporcional a su valor esperado (f_i/F). En la selección proporcional, una vez generada la *pool*, escogíamos aleatoriamente dos individuos para su reproducción, suponiendo que en la misma estructura teníamos de forma implícita una probabilidad de selección acorde al valor esperado de cada individuo. Para la selección por ruleta perfeccionamos este mecanismo: en lugar de generar la *pool*, se calcula el valor f_i/F, se toma un número aleatorio r entre 0 y F, y se recorre la población, calculando la suma acumulada de los valores esperados de cada individuo hasta que se sobrepase la cantidad r. El primer individuo que lo consiga pasa a formar parte de la selección. Repetimos este procedimiento tantas veces como individuos necesitemos. El proceso se ve recogido en el algoritmo 3 (para un estudio más detallado puede consultarse [SD07]).

Algoritmo 3: Roulette Wheel Algorithm

$\mathsf{population} = \{m_i \mid i \leq N\}$
$F \leftarrow \sum_{i=1}^{N} \texttt{fitness}(m_i)$
$r \leftarrow \texttt{rand}(0, F)$
$threshold \leftarrow 0$
$i \leftarrow 0$
while $threshold \leq r$ **do**
  $i \leftarrow i + 1$
  $threshold \leftarrow threshold + \frac{m_{i-1}}{F}$
end
return m_i

3.1.4. Selección por rango

La selección por rango difiere de los métodos anteriores en que no tiene en cuenta el valor concreto de *fitness* de cada individuo, sino el orden que este genera en la población. El método es muy simple y se han dado varias implementaciones distintas (en [SD07] se proponen dos diferentes). La idea general es seleccionar aleatoriamente un par de individuos de los cuales escogeremos el que mejor *fitness* tenga para la selección. El proceso se repite tantas veces como individuos se requieran. A este proceso se le puede considerar un caso especial de la Selección por Torneo, en la que se aplica el mismo esquema salvo que se escoge una cantidad n de individuos aleatorios en vez de 2.

3.1.5. Selección estocástica

El último método que estudiamos para la selección es la estocástica. Similar a la ruleta y a la selección proporcional, este mecanismo difiere de los anteriores en que en cada selección asegura que los individuos estén equiespaciados en la población. Los primeros métodos simulaban el hacer girar una ruleta, asumiendo que la probabilidad se distribuía acorde al valor esperado de *fitness* de cada individuo, pero haciendo esto en tiradas diferentes no podemos asegurar que sean seleccionados individuos con una distancia entre sí lo suficientemente grande para asegurar diferencias notables entre ellos, esto es, la probabilidad de que las distancias $\delta_{i,j}$ entre cualesquiera m_i, m_j individuos de la selección

(en términos de diversidad) esté acotada por una cantidad $0 < \delta_{i,j} < \epsilon$ arbitrariamente pequeña, es no nula. Es por ello que la selección estocástica recoge todos los miembros para la reproducción en una misma pasada (figura 2), asegurando $\frac{N}{N_s} \leq \delta_{i,j}$, siendo N la cantidad de individuos en la población, N_s el número de estos a seleccionar y $\delta_{i,j}$ la distancia entre cualesquiera m_i, m_j de ellos. Los pasos quedan reflejados en el algoritmo 4.

Algoritmo 4: Stochastic Universal Sampling

$\mathsf{population} = \{m_i \mid i \leq N\}$
$N_s \leftarrow$ num. indiv.
$step \leftarrow \frac{N}{N_s}$
$start \leftarrow \mathtt{rand}(0, step)$
return $\{m_{start}, m_{start+step}, ..., m_{start+(N-1)*step}\}$

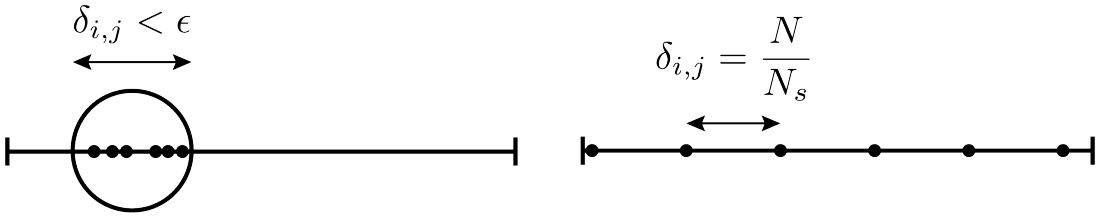

Figura 2: Detalle selección estocástica

3.2. Reemplazo

Una vez determinado el método de selección, el siguiente paso es idear una estrategia similar para posicionar los nuevos individuos en la población. Un concepto muy importante sobre este aspecto es el elitismo, es decir, individuos de la población que quedarán fijos para la siguiente. Es evidente que buscaremos que dichos individuos se encuentren entre los mejor adaptados, pues reemplazarlos a todos no garantiza una mejora de la población a nivel de adaptación. De este modo, consideraremos siempre el elitismo en los algoritmos, siendo posible determinar qué cantidad de elitistas produce mejores resultados en el algoritmo. Habitualmente, se usa el modelo en el que solo tomamos un elitista, lo cual se conoce como "*golden cage model*"[AWWB09], ya que permite asegurar que

$$\text{mín}\{fitness(m_i) \mid m_i \in P(t)\} \leq \text{mín}\{fitness(m_i) \mid m_i \in P(t-1)\}$$

Es decir, es el modelo minimal que asegura una evolución favorable de la población en términos de adaptación.

3.2.1. Reemplazo total

Nos referiremos al reemplazo total (o reemplazo estándar) cuando sustituyamos todos los individuos de la población, salvo quizá un número fijo de elitistas.

3.2.2. Reemplazo por algoritmo *qNNR*

El reemplazo por el algoritmo *qNNR* (*q-nearest neighbour replacement*, [LK08]) trata de evitar la convergencia prematura y la pérdida de capacidad de explorar otros extremos en la función de coste. Dado que *a priori* no se conoce el comportamiento de la función a minimizar, esta puede constar de varios mínimos locales, quizá todos ellos globales, por lo que el algoritmo deberá ser capaz de no limitarse a la búsqueda de solo uno de ellos. Este fenómeno en el que las soluciones quedan atrapadas en un entorno de un mínimo se conoce como *niching* (en términos biológicos como nicho de población). Para evitarlo, cuando generemos un nuevo individuo, lo reemplazaremos por el peor en un entorno de tamaño q, de modo que escogiendo adecuadamente este valor podremos conseguir soluciones fuera del entorno del mínimo, lo cual permitirá que el algoritmo acabe explorando el resto. Detallamos los pasos en el algoritmo 5.

Algoritmo 5: q-Nearest Neighbour

```
population = {m_i | i ≤ N}
t ← current iteration
q(t) ← size of neighbour set
K ← number of replacements
for 1 ≤ s ≤ K do
    m_k, m_l ← selection(population)
    m ← crossover(m_k, m_l)
    mutate(m)
    calculate {d(m, m_j) | j = 1, ..., N}
    sort (population) from distances
    P̃ ← {m_1, ..., m_q(t)} first q(t) elements of population
    m̃ ← {m_j | fitness(m_j) ≤ fitness(m_i), m_i, m_j ∈ P̃, i, j = 1, ..., q(t), i ≠ j}
    if fitness(m) < fitness(m̃) then
        m̃ ← m
    end
end
```

En nuestro caso hemos optado por una variante dinámica en la cual el tamaño $q(t)$ es adaptado en cada iteración del siguiente modo:

$$q(t) = q_{min} + \left(\frac{e^{\frac{t}{T_{max}}} - 1}{e - 1} \right)^{\alpha} (q_{max} - q_{min})$$

Este mecanismo dinámico nos permite en generaciones tempranas, al tener un valor q pequeño, mejorar localmente las soluciones de modo que los posibles nichos de población vayan quedando más definidos. A medida que avanza el algoritmo y el valor q va creciendo, el reemplazo por el peor individuo en el entorno se hace a nivel más global, consiguiendo de este modo una mejor localización del mínimo global (si fuese único), y un reparto entre ellos en caso de no serlo.

En resumen, el algoritmo *qNNR* con la variante dinámica es capaz de localizar individuos en los nichos para generaciones tempranas y discernir los mínimos globales para generaciones avanzadas.

Hasta ahora, los métodos que hemos descrito tienen complejidad lineal en el tamaño de la población (tanto selección como reemplazo), pero este último tiene coste cuadrático $O(n^2)$, lo cual ha de tenerse en cuenta a la hora de su aplicación.

3.3. Cruce

Fijados los métodos de selección y reemplazo, resta determinar cómo la información genética de los predecesores es combinada e incluida en los nuevos individuos. El mecanismo de cruce genera un nuevo individuo con características similares a las de un conjunto dado de ellos, los predecesores, mediante la aplicación de los operadores de cruce. Básicamente imita el proceso reproductivo que poseen las especies en biología, por lo que pueden describirse varios tipos, por ejemplo, reproducción asexual donde el conjunto de predecesores es unitario, sexual cuando es binario, o podemos considerar cualquier otro conjunto. De este operador se espera que sea capaz de transmitir correctamente la información de los predecesores a los descendientes, y que además conserve su estructura. Es aquí cuando nos encontramos con un impedimento: si nuestro problema es de optimización será necesario que nuestras soluciones sean factibles, de modo que cabría esperar que exista un operador que transforme soluciones factibles en soluciones factibles. Raramente estos operadores serán viables desde el punto de vista computacional pues tendrán una alta complejidad, lo cual anulará toda validez a la hora de aplicar algoritmos genéticos.

3.3.1. Cruce uniforme

El operador de cruce uniforme toma dos individuos para generar uno nuevo, de modo que cada componente del nuevo venga dada por una elección de las correspondientes componentes de los antecesores (para cada coordenada se hace una elección aleatoria entre los predecesores, como se ejemplifica en la figura 3).

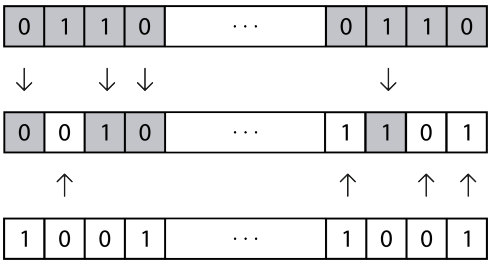

Figura 3: Cruce uniforme

3.3.2. Cruce por N puntos

El cruce por N puntos toma N posiciones aleatorias en el vector solución. De modo alternado, en cada sección que determinan dichos puntos se copia el contenido genético de cada predecesor. En este trabajo hemos implementado el cruce por uno y dos puntos. Hay que destacar que el método anterior es un caso particular de este, donde N se corresponde con la longitud del ADN. Es de esperar que, a medida que N se aproxime a la longitud del ADN, las diferencias entre ambos métodos disminuyan.

3.3.3. Cruce por operador AND

El cruce por operador AND hace uso de la naturaleza booleana de nuestra representación, de modo que cada coordenada del nuevo individuo viene determinada por la aplicación del operador a los predecesores. Cabe preguntarse por qué no consideramos un cruce por el operador OR. Si bien puede esperarse que OR produzca con mayor probabilidad soluciones factibles, los resultados previos han mostrado que su rendimiento es demasiado pobre, por lo que hemos decidido no incluirlo en el estudio.

3.4. Mutación

Con el objetivo de simular más fielmente la selección en el ámbito biológico, los algoritmos genéticos se han visto obligados a considerar el concepto de mutación. No solo por imitación, la mutación cobra sentido cuando observamos que, al partir de una población inicial aleatoria, nada impide que dicha población esté representada por genes de la forma $(x_1, ..., x_{n-1}, 1)$. Dado que a partir de estos genes haremos actuar los operadores de selección, cruce y reemplazo, puede observarse que la última posición de todos los nuevos individuos obtenidos seguirá siendo 1. Si la solución óptima a nuestro problema tuviese como última coordenada un 0, estaríamos condenados al fracaso desde la generación inicial. Este ejemplo que resulta más visual puede aplicarse a situaciones no tan evidentes, pero que igualmente limitan drásticamente la capacidad de estos algoritmos, por ejemplo, si en cierto tiempo t la población queda estancada en un extremo local de la función a optimizar y la población carece de suficiente diversidad genética para prosperar. Este último término, diversidad genética, es una de las razones por las que tenemos en consideración la mutación. Un cambio aleatorio en la representación del ADN de cada individuo evita que se den situaciones como las anteriores, al menos en un tiempo lo suficientemente grande. Dicho esto, en este trabajo consideramos dos tipos de mutación que pasaremos a describir.

3.4.1. Mutación por intercambio

La mutación por intercambio, como su propio nombre indica, toma dos posiciones aleatorias del ADN de un individuo y las intercambia. Puede observarse que para que este mecanismo genere un cambio real ha de darse la situación de que en las dos posiciones el ADN contenga símbolos distintos.

3.4.2. Mutación aleatoria

En la mutación aleatoria se toma una posición arbitraria del ADN y se cambia su valor. Tanto en la anterior mutación como en esta, prefijamos una probabilidad de que ocurra un cambio con valores típicamente bajos que se asemejen a la realidad.

4. Caso de estudio

Como se ha mencionado ya en esta memoria, uno de los campos donde es más lícito el uso de algoritmos genéticos es en el campo de la optimización con restricciones. A continuación, presentamos un problema típico en el campo del *marketing* que reúne todas las condiciones para ser resuelto por estas técnicas evolutivas.

4.1. Introducción al problema y su representación

Los dos problemas que presentamos, APO (*Advertisement Placement Optimization*) y PAPO (*Probabilistic Advertisement Placement Optimization*), variantes de un mismo problema más general, tratan de resolver la cuestión de cómo situar anuncios en distintos medios. El problema es el siguiente: se desea encontrar un posicionamiento óptimo (de mínimo coste) de un anuncio en distintos medios, de modo que para cada medio obtengamos un nivel de visitas por encima de una cantidad prefijada. Más aún, tendremos en consideración distintos segmentos de población sobre los que diferirán las visitas alcanzadas (por ejemplo, es evidente que para un cierto medio, no se reciben la misma cantidad de visitas de un segmento de población que represente a un estudiante de secundaria que de otro que represente a personas interesadas en la bolsa). Cómo medimos el nivel de visitas mínimas que se desean alcanzar y cómo expresamos esa dependencia en términos de los medios y segmentos de población determina los dos tipos de problemas que mencionábamos anteriormente. Una primera aproximación al problema es considerar que para un par de medio/segmento la probabilidad es fija (método fácilmente aplicable pues los medidores de audiencia proporcionan este tipo de estadísticas directamente). De este modo, podemos expresar las restricciones de modo que la suma de los valores medios de visitas en cada medio para cierto segmento sea mayor o igual que cierta cantidad. Dado que la restricción hace referencia a una cantidad media de visitas, no es posible determinar si estas provienen de múltiples individuos o si, por el contrario, todas ellas se deben a uno solo. Esta aproximación, si bien es más directa en el sentido del uso de los datos de los que se disponga, tiende a ser menos precisa dando lugar a soluciones que *a priori* no se esperan obtener.

La otra aproximación del problema solventa la anterior desventaja, cambiando las restricciones para que hagan referencia a la probabilidad de que una persona en el segmento considerado vea el anuncio al menos una vez. Esto podemos conseguirlo introduciendo dependencias entre los medios donde se puede posicionar el anuncio y las probabilidades de visualización de cada segmento. De dichas dependencias esperamos poder especificar, por ejemplo, que la probabilidad de que una persona correspondiente a un segmento venga dada por la unión de sucesos, o que dicha probabilidad venga dada por sucesos independientes (ver el medio M es incompatible con el medio N). Con estas dependencias queremos poder expresar, por ejemplo, la existencia de medios incompatibles: una persona no puede visualizar dos medios simultáneamente, por ejemplo, 'documental' y 'película' en la misma franja horaria, o expresar también que una persona pueda ver indistintamente ciertos eventos; todo ello en cada segmento de población, entre los cuales dichas dependencias variarán. Es por esto que consideraremos estas dependencias en términos de independencia, exclusión e inclusión de eventos. Este último es muy relevante en términos de coste, pues puede ser mucho menos costoso posicionar el anuncio en el medio M que en el N, y podemos conseguir la misma audiencia si las personas que ven el medio N también ven el M. Obsérvese que expresar las restricciones en estos términos puede dar

lugar a dependencias cíclicas sobre todo en el último caso. Para evitarlo, los construiremos en forma de árbol, obligando a que cada evento aparezca una única vez en cada término.

4.2. Formalización

Las dos variantes del problema aquí tratadas (APO, PAPO) se asemejan en estructura y se diferencian únicamente en el cómputo de las restricciones. Ambos problemas están expuestos formalmente en [RRR16]; por motivos de claridad repetimos su exposición en esta sección.

Consideremos, dadas $m, n \in \mathbb{N}$, el conjunto de medios $M = (M_1, ..., M_n)$ donde podremos situar el anuncio, el conjunto de segmentos de población considerados $S = (S_1, ..., S_m)$ sobre los cuales actuarán las restricciones, el vector de visitas mínimas $b = (b_1, ..., b_m)$ para cada segmento, y el vector de costes $c = (c_1, ..., c_n)$.
En ambas variantes, el espacio de soluciones viene dado por vectores de la forma $x = (x_1, ..., x_n)$ con cada $x_i \in \{0, 1\}$, siendo $x_i = 1$ posicionar el anuncio en el medio M_i y $x_i = 0$ no posicionarlo.

4.2.1. APO

Dadas m, n, el conjunto de medios M, el de segmentos S y los vectores de restricciones b y de costes c, el problema APO trata de minimizar la función $c^T x$ sometido a las restricciones $Ax \geq b$, siendo A la matriz en la cual cada componente a_{ij} representa la media de vistas registradas para el segmento de población S_i en el medio M_j.

Se tiene el siguiente teorema [RRR16, Theorem 1]:

Teorema 1: APO $\in$ Log–APX–hard.

Este resultado es lo que justifica la elección de los algoritmos genéticos en la resolución de estos problemas; dado que computacionalmente es muy costoso encontrar la solución óptima, optamos por un algoritmo que aproxime dicha solución. Para el problema APO en concreto, no existe ningún algoritmo polinómico que la aproxime (dado que pertenece a Log–APX–hard) por lo que es legítimo el uso de un algoritmo genético a tal propósito.
Dada la forma lineal que presenta el problema, cabe preguntarse si es acertado plantearlo como uno de programación lineal con variables binarias, lo cual no es muy acertado pues el Simplex (y alguna de sus variantes), para ciertos problemas ([KSZ09]) se ha demostrado que tiene coste exponencial, lo cual difiere poco de realizar una búsqueda exhaustiva.

4.2.2. PAPO

En la exposición de este problema hemos mencionado que las restricciones se expresaban en forma de árbol. La siguiente gramática formaliza esa idea:

$$\begin{aligned} A &\longrightarrow \mathsf{indep}(A, ..., A) \mid \mathsf{excl}(A, ..., A) \mid \mathsf{incl}(A, B) \mid B \\ B &\longrightarrow e_1 \mid ... \mid e_n \end{aligned} \tag{1}$$

donde $\mathsf{indep}(A, ..., A)$ representa la independencia de términos, $\mathsf{excl}(A, ..., A)$ la exclusión e $\mathsf{incl}(A, B)$ la inclusión.

Puesto que la gramática no restringe las apariciones de eventos[3], forzaremos que esto ocurra definiendo $\mathcal{WF}(t)$, que representará los términos *bien fundados*:

$$\mathcal{WF}(t) = \begin{cases} \bigwedge_{k=1}^{m} \mathcal{WF}(t_k) & t = \mathsf{indep}(t_1, ..., t_m) \ \wedge \ \mathsf{ev}(t_i) \cap \mathsf{ev}(t_j) = \emptyset \ \forall i \neq j \\ \bigwedge_{k=1}^{m} \mathcal{WF}(t_k) & t = \mathsf{excl}(t_1, ..., t_m) \ \wedge \ \mathsf{ev}(t_i) \cap \mathsf{ev}(t_j) = \emptyset \ \forall i \neq j \\ \mathcal{WF}(t_1) \wedge \mathcal{WF}(t_2) & t = \mathsf{incl}(t_1, t_2) \ \wedge \ \mathsf{ev}(t_1) \cap \mathsf{ev}(t_2) = \emptyset \\ 1 & t = e_i \\ 0 & t \neq e_i \end{cases}$$

donde $\mathsf{ev}(t)$ representa los eventos del término t y se define de forma recursiva de un modo análogo.

A partir de (1), podemos calcular la probabilidad de ver el anuncio bajo el término restrictivo t, el conjunto de probabilidades P para cada evento y cada asignación de medios x como sigue:

$$\omega(t, P, x) = \begin{cases} 1 - \prod_{k=1}^{m} (1 - \omega(t_k, P, x)) & t = \mathsf{indep}(t_1, ..., t_m) \\ \sum_{k=1}^{m} \omega(t_k, P, x) & t = \mathsf{excl}(t_1, ..., t_m) \\ P(e_i) & t = \mathsf{incl}(t_1, e_i) \ \wedge \ x_i = 1 \\ \omega(t_1, P, x) & t = \mathsf{incl}(t_1, e_i) \ \wedge \ x_i = 0 \\ P(e_i) & t = e_i \ \wedge \ x_i = 1 \\ 0 & t = e_i \ \wedge \ x_i = 0 \end{cases}$$

Nótese que el hecho de añadir más expresividad en las restricciones no añade computacionalmente complejidad, como puede verse en el siguiente teorema [RRR16, Theorem 2]:

Teorema 2: PAPO $\in$ Log–APX–hard.

[3]Recordemos que no restringir las apariciones de eventos puede dar lugar a dependencias cíclicas como, por ejemplo, $\mathsf{incl}(A, B), \mathsf{incl}(B, C)$ e $\mathsf{incl}(C, A)$.

4.3. Peculiaridades

Una vez formalizado el problema, hemos de reflejar todos los conceptos de un algoritmo genético en términos del problema específico. La representación será binaria (de longitud fija) como ya se ha mencionado. Una cuestión importante es la factibilidad. Dado que buscamos soluciones a un problema con restricciones, sería deseable que nuestro algoritmo fuese capaz de trabajar con estas; dicho de otro modo, que de soluciones factibles genere nuevas soluciones factibles. Un avance en este aspecto puede consultarse en [LGT08], donde se expone una adaptación de los algoritmos PBS (*pseudo-boolean solvers*) para la preservación de la factibilidad mediante el *crossover* y mutación. La estrategia seguida en este trabajo es penalizar a los individuos no factibles mediante la función *fitness*, que vendrá dada por el coste de la solución x, $c^t x$, más una penalización por cada restricción no cumplida:

$$fitness(x) = \begin{cases} c^t x & x\ feasible \\ c^t x + M(x) & otherwise \end{cases}$$

siendo

$$M(x) = \overline{C} \left(\sum_{j \in \mathcal{R}} \frac{b_j - \omega(t_j, P_j, x)}{b_j} \right)$$

en el caso del problema PAPO, y de modo similar

$$M(x) = \overline{C} \left(\sum_{j \in \mathcal{R}} \frac{b_j - (Ax)_j}{b_j} \right)$$

para el problema APO. En ambos casos, $\mathcal{R}$ representa el conjunto de restricciones no satisfechas y $\overline{C} = n \max\{c_k \mid k = 1, ..., n\}$.

Otro aspecto relacionado con la factibilidad lo encontramos en la inicialización. Si bien el algoritmo toma como generación cero una población aleatoria, sería deseable que la gran mayoría de individuos (por no decir todos) fuesen factibles. Nuestra estrategia es inicializar todos los miembros al vector de componentes 1 y aplicarles una mutación, de modo que haya una probabilidad muy alta de que los individuos sean factibles. Además, fijamos uno de ellos al vector 1, permitiéndonos de este modo, comprobar en la primera iteración del algoritmo si el problema posee solución. Como se propone en [RRR16], podemos incluir en la población 0 la solución del algoritmo heurístico. Las palabras de Wolpert y Macready en su artículo *"No free lunch theorems for optimization"* resaltan la importancia del conocimiento previo del problema y su incorporación al comportamiento del algoritmo con el fin de obtener mejoras en su rendimiento:[4]

> " *... results also indicate the importance of incorporating problem-specific knowledge into the behavior of the algorithm."* [WM97]

[4]Más adelante se expondrán los resultados del artículo, así como el teorema fundamental.

4.4. Heurísticas

Para la resolución de los problemas expuestos, se han desarrollado técnicas heurísticas para aproximar sus soluciones (en [RRR16] pueden consultarse detalladamente dichas heurísticas). Al presentar un marco de referencia sobre el cual comparar la efectividad de los algoritmos genéticos, hemos incluido en el estudio los resultados obtenidos por estas heurísticas. En líneas generales, estos métodos surgen de modo natural de los teoremas que muestran la clase de complejidad a las que pertenecen (considerando reducciones de otros problemas bien conocidos y para los cuales se conocen heurísticas, adaptando estas a los problemas aquí tratados, proporcionando también cotas de aproximación).

5. Análisis

En esta sección mostraremos los resultados obtenidos de la aplicación del algoritmo genético con los operadores anteriormente descritos a los problemas APO y PAPO. En la medida de lo posible, hemos tratado de aplicar los criterios para medir la eficacia de estos algoritmos descritos en [LA11].

Para cada configuración de parámetros del algoritmo (mutación, cruce, selección y reemplazo) hemos realizado un test en el que se han tomado los datos del mejor individuo en cada iteración, a lo largo de las 250 iteraciones fijadas para el algoritmo, con una población inicial de 200 y 20 elitistas. Este proceso se ha repetido 100 veces por cada test.

Con el fin de localizar qué operadores son más apropiados para los problemas particulares ya descritos, hemos optado por empezar generando una colección de instancias aleatorias del problema con diferentes combinaciones de parámetros, como pueden ser la cantidad de medios, segmentos y profundidad del árbol de restricciones (en el caso PAPO). Hemos sometido una instancia en particular a todas las posibles combinaciones de los operadores que forman el algoritmo genético, obteniendo un total de 80 distintas, con el fin de identificar un grupo de configuraciones que mejor resuelva los problemas.

La estrategia por la que hemos optado para realizar el análisis ha sido comparar los resultados de cada método, buscando evidencia estadística para determinar diferencias. Puesto que disponemos de un amplio abanico de ejecuciones (dadas por todas las combinaciones de operadores), hemos aplicado reiteradamente el test de Kruskal-Wallis, un test no paramétrico (no supone normalidad en los datos) que nos permite realizar comparaciones basadas en grupos, sin que sea necesaria la misma cantidad de datos por cada uno de estos. Como criterio para determinar qué operador se adapta mejor al problema, hemos tomado el que mayor diferencia produce en dichos test. Consideramos esta estrategia lícita pues es la que asegura mayor influencia del método seleccionado frente al resto del grupo.

Se han sometido a análisis los datos obtenidos agrupados por método de cruce, mutación, selección y reemplazo. Una primera conclusión es que el método de cruce uniforme obtiene, en general, mejores resultados que el resto, siendo a su vez esta comparación la que mayor relevancia presenta, es decir, el método de cruce es el que más influye en el resultado final del algoritmo. A lo largo de todo el análisis, hemos tomado como umbral $p = 0{,}05$. El test de Kruskal-Wallis confirma una diferencia entre los operadores de cruce con $p = 1{,}2178e - 320$ para el caso PAPO y $p = 0$ para APO. Los resultados se reflejan en la figura 4[5].

Esto se debe principalmente a que es el método que más uniformemente reparte el contenido genético de cada individuo, ya que no solo lo distribuye entre los dos progenitores, sino que lo hace además a lo largo de la longitud del nuevo individuo. Puede observarse que, aun siendo el cruce uniforme el que mejores soluciones encuentra (y de modo más estable, es decir, con menos variación), el cruce por dos puntos también es capaz de encontrar buenas soluciones. Es más, aumentando el número de puntos en dicho cruce el operador converge al cruce uniforme, por lo que los resultados obtenidos son acordes al modelo teórico.

[5]Como puede observarse, el operador AND en la figura alcanza un valor 0. Esto es debido a la implementación, puesto que decidimos asignar un valor *fitness* 0 a las soluciones que no fuesen factibles, de modo que fuesen fácilmente distinguibles del resto.

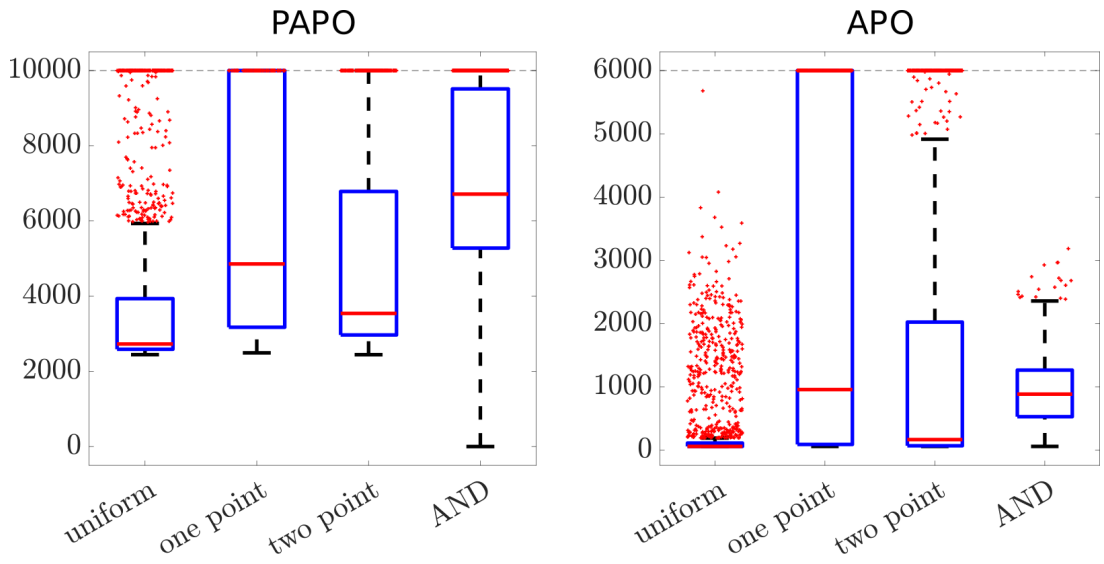

Figura 4: Comparación método de cruce

Fijado como grupo de estudio el correspondiente al cruce uniforme, se ha repetido el análisis y se ha llegado a la conclusión de que el método de reemplazo es el que mayor influencia presenta en los resultados; el algoritmo *qNNR* proporciona mejores soluciones frente al reemplazo estándar. Este hecho queda respaldado, de nuevo, por el test de Kruskal-Wallis, el cual determina una diferencia significativa con un p-valor 3.9139e-69 (PAPO) y 2.2475e-86 (APO). El resultado puede verse en la figura 5. Un análisis posterior revela que, para ambos problemas, el método de selección por rango es con diferencia el peor entre los considerados. El grupo restante sigue presentando diferencia para PAPO ($p = 0{,}0069$), pero en el caso APO tenemos $p = 0{,}1625$.

Hasta ahora tenemos fijados el operador de cruce (uniforme) y de reemplazo (*qNNR*) y aún libres la selección y sin determinar la mutación. Con respecto a esta última, presenta comportamientos opuestos para los dos problemas: en PAPO obtiene mejores resultados el intercambio y en APO la aleatoria. Esto puede explicarse por la no linealidad que presenta PAPO, por lo que siendo más difícil conservar la factibilidad, las soluciones que no lo sean son penalizadas bruscamente, de modo que una mutación por intercambio tiene menos probabilidad de modificar drásticamente el valor *fitness* que una aleatoria, pues la primera no aumenta el número de elecciones y la aleatoria lo hace con probabilidad 1/2. Una vez considerada la mutación correspondiente para cada problema, llegamos al punto en que no existe evidencia que determine diferencias entre los operadores de selección.

Una vez fijado el grupo de configuraciones que mejor aproximan la solución, pasamos a estudiar en detalle los aspectos más destacados de estos métodos. Dado que todos son capaces de encontrar soluciones aceptables y que entre ellos no presentan diferencia significativa, tendremos en consideración otros aspectos de los algoritmos para seleccionar el más adecuado, por ejemplo, el tiempo de ejecución o la velocidad de convergencia.

La figura 6 muestra la evolución del valor *fitness* de los métodos seleccionados a lo largo de las 250 iteraciones. Como puede observarse, todos los métodos presentan un comportamiento similar. Además, para el problema APO puede apreciarse una convergencia ligeramente más rápida debido, principalmente, a la no linealidad de PAPO.

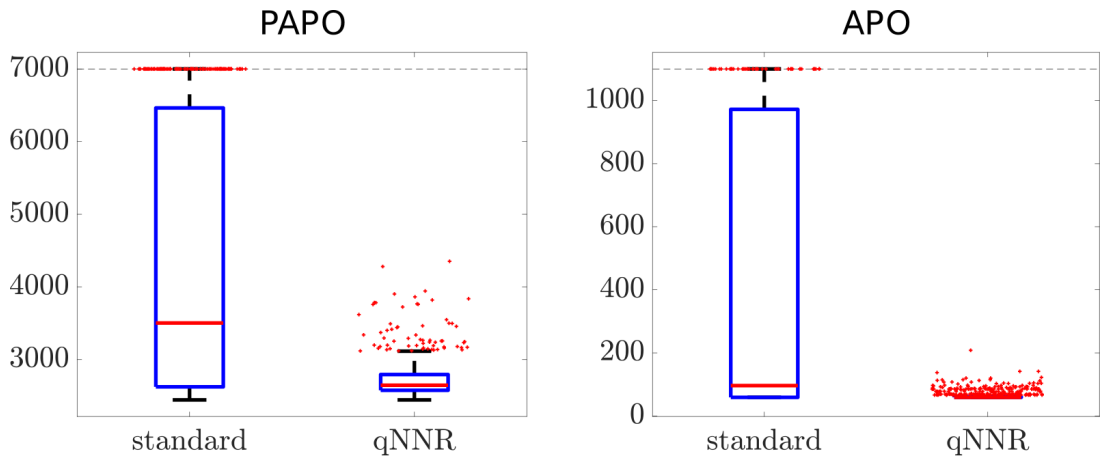

Figura 5: Comparación grupos *standard* - *qNNR*

Con respecto a la factibilidad, hemos de destacar que para APO todos los individuos son factibles y para PAPO también lo son con alto porcentaje (97-100 %). Puede concluirse que los operadores seleccionados junto a la inicialización realizada y las penalizaciones del valor *fitness* para individuos no factibles operan acorde al espacio de búsqueda.

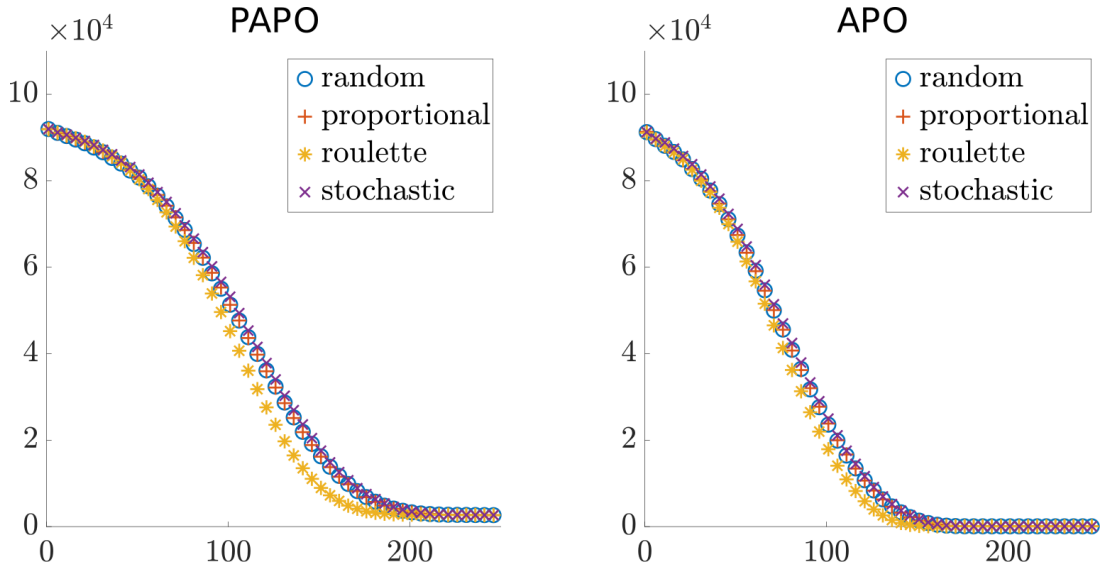

Figura 6: Evolución *fitness*

Como se ha mencionado anteriormente, la medida de diversidad por la que hemos optado es una métrica basada en la distancia de *Hamming*. La evolución de esta a lo largo de la ejecución se muestra en la figura 7.

Puede observarse que todos los métodos presentan baja diversidad. Recordemos que, con esta métrica, una buena diversidad se traduce en valores lo más grandes posibles (siempre acotados por la longitud del ADN, l, en este caso 200). Ciertamente, no podemos esperar valores cercanos a esta cantidad, dado que implicaría que todos los miembros de la población fuesen completamente distintos, situación imposible a menos que nuestra población constase únicamente de dos miembros. Además, nos encontramos por debajo del umbral $l/2$ que indicaría una distribución uniforme. Es de notar, sin embargo, que

con el fin de mejorar la factibilidad, hemos partido de soluciones muy similares entre sí [6], lo cual explica la baja diversidad al comienzo del algoritmo; por otra parte, es lógico que a medida que avanza la ejecución, dicha cantidad también disminuya pues la mayoría de los individuos tenderán a parecerse más entre sí.

Aumentando las iteraciones (en el caso de prueba doblando la cantidad), podemos observar mejoras para PAPO (con $p = 1{,}0614e{-}30$) y APO ($p = 0{,}0018$). Aunque la evolución del valor *fitness* sigue la misma tendencia al aumentar las iteraciones, nos encontramos frente a una mejora del 3 % para PAPO, y del 1 - 2 % en APO.

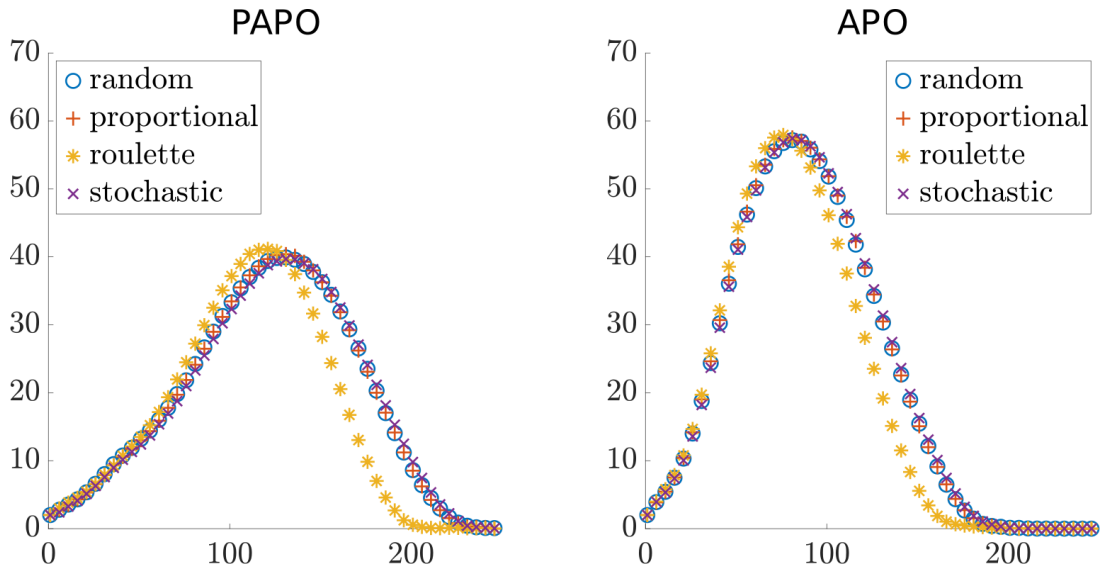

Figura 7: Evolución de la diversidad genética

Del mismo modo, hemos aumentado los tamaños[7] de población y *pool*. Los resultados obtenidos, mostrados en la tabla 1, muestran claramente que el grupo correspondiente a una población de 500 mejora los obtenidos con un tamaño de 200, mientras que el tamaño de la *pool* no es relevante para los resultados. Puede observarse que esta tendencia se cumple para PAPO, dado que en APO se presentan diferencias para los operadores de selección proporcional y selección estocástica.

Grupo Op.	Población 200				Población 500				Sin grupos
	Rand.	Prop.	Roulet.	Stoch.	Rand.	Prop.	Roulet.	Stoch.	
APO	0.1223	0.9741	0.2221	0.5511	0.1563	$6{,}69e-4$	0.4083	$7{,}46e-7$	$8{,}89e-25$
PAPO	0.4243	0.5469	0.1034	0.8278	0.5936	0.2154	0.9744	0.2668	$8{,}28e-103$

Tabla 1: p-valores para los diferentes tamaños de población/*pool*

[6] La inicialización se realizaba estableciendo todas las componentes del ADN a 1 y posteriormente una mutación.

[7] población: 200, 500; *pool*: 200, 500.

Con el fin de asegurar la independencia entre los resultados obtenidos (mejores configuraciones del algoritmo para los problemas específicos) y las dimensiones específicas del problema (medios, segmentos y profundidad del árbol de restricciones), hemos aplicado las configuraciones determinadas anteriormente a problemas con distintas dimensiones cuyos resultados se recogen en las tablas 2, 3 y 4.

media		50				100				200			
segmentos		1	5	10	20	1	5	10	20	1	5	10	20
AG	mejor	385	507	1493	1331	540	718	1298	1634	111	1694	2928	3830
	media	387	508	1501	1343	545	728	1309	1661	117	1741	3015	3937
heurística		385	579	1790	1491	570	718	1321	1791	111	1802	3009	4066

Tabla 2: Resultados para APO

media		50				100				200			
segmentos		1	5	10	20	1	5	10	20	1	5	10	20
AG	mejor	75	808	2136	3298	704	2079	3558	3918	63	1595	1594	5259
	media	75	808	2298	3363	706	2098	3862	4080	76	1958	1919	6522
heurística		687	2218	7634	8289	704	7859	14107	11472	63	3194	4279	14173

Tabla 3: Resultados para PAPO, $depth = 5$

media		50				100				200			
segmentos		1	5	10	20	1	5	10	20	1	5	10	20
AG	mejor	178	715	1305	1791	53	370	1213	2093	154	941	1455	2443
	media	178	796	1356	1994	53	379	1248	2700	192	1032	1794	2649
heurística		178	2469	7801	6158	617	2249	2975	5325	968	2443	9577	5812

Tabla 4: Resultados para PAPO, $depth = 10$

Los datos remarcados indican las instancias para las que hemos detectado diferencias significativas en los métodos utilizados. Ya que dichas diferencias se presentan en un 16-25 % de los casos, cabe preguntarse si estas se deben a la instancia del problema en particular o bien a la propia estructura de este.

Tras generar más instancias de los problemas, se ha detectado que algunas de ellas sí la presentan. Los datos que hemos recogido se exponen en las tablas 5 y 6.

Los porcentajes están expresados sobre 10 problemas distintos para cada dimensión. Dadas las limitaciones a nivel de capacidad de cómputo que disponemos, no ha sido posible recolectar esta información para un mayor número de problemas.

En este punto, debe señalarse que, si bien los resultados para PAPO no presentan una clara tendencia en cuanto a las diferencias entre métodos, para APO puede observarse que el porcentaje de instancias que presentan diferencias crece a medida que lo hace la complejidad del problema, es decir, a medida que los segmentos y medios aumentan. Esto puede ser un claro indicativo de que el rendimiento del algoritmo genético está intrínsecamente ligado a la estructura de este.

media	50				100				200			
segmentos	1	5	10	20	1	5	10	20	1	5	10	20
	60	30	30	60	50	80	70	80	90	90	90	90

Tabla 5: Porcentaje de instancias con diferencia significativa para APO

media		50				100				200			
segmentos		1	5	10	20	1	5	10	20	1	5	10	20
profundidad del árbol	5	0	20	20	20	20	20	0	20	20	40	0	30
	10	0	20	20	30	0	0	20	30	30	10	10	10

Tabla 6: Porcentaje de instancias con diferencia significativa para PAPO

Con el fin de dar una interpretación a los resultados expuestos, introducimos los siguientes conceptos:

Consideremos las mediciones de un algoritmo sobre cierto problema, denotadas por d_m donde m representa la cantidad de mediciones tomadas; sean $f \in \mathcal{F}$ funciones de coste a optimizar, y sea $\{a_i\}_{i \in I}$ el conjunto de algoritmos aplicados. El teorema NFL (*no-free-lunch* [WM97]) asegura que, dados a_1, a_2 algoritmos cualesquiera, se cumple que

$$\sum_f P(d_m|f, m, a_1) = \sum_f P(d_m|f, m, a_2)$$

El resultado que nos ocupa es un corolario del anterior, que en palabras de los autores en el artículo original se expone:

> "*... for any performance measure* $\Phi(d_m)$*, the average over all* f *of* $P(\Phi(d_m)|f, m, a)$ *is independent of* a*. The precise way that the sample is mapped to a performance measure is unimportant.*" [WM97]

Es decir, la distribución de las medidas realizadas sobre un problema (o instancia) no dependen del algoritmo en concreto, por lo que si un algoritmo presenta mejores resultados para cierto problema, deberá presentar otros peores para otras instancias. Este resultado puede explicar los datos recogidos en las tablas 5 y 6, ya que el grupo de configuraciones del algoritmo que habíamos seleccionado presenta diferencias estadísticamente significativas cuando se aplica a otras instancias del problema. Hemos de incidir en que, si bien se dan estas diferencias, no podemos deducir que estas estén provocadas por un algoritmo que rinda mejor, sino que pueden ser debidas también a que uno de ellos presente un rendimiento muy inferior. Este resultado no implica que el análisis realizado anteriormente sea en vano ya que con él queda demostrada una evidente mejora de estos algoritmos frente a las técnicas heurísticas.

Con respecto al alcance y aplicabilidad del teorema NFL, hemos de decir que existen varios autores que discuten la practicidad de los teoremas para problemas reales fuera del marco absolutamente teórico; por ejemplo, en [RR02] se expone un capítulo dedicado al tema con un enfoque centrado en el contexto de los algoritmos genéticos donde se hace referencia a dicha aplicabilidad.

En nuestro caso, no podemos asegurar que los teoremas NFL sean la única justificación de los resultados, pues si bien entran dentro de las hipótesis, estas son mucho más

generales, por lo que dichos teoremas pueden estar haciendo referencia al hecho de que los algoritmos genéticos aquí considerados resuelvan bien los problemas APO y PAPO, pero aplicando dicho algoritmo a otro problema sus resultados sean pobres, cumpliéndose así los resultados de NFL.

6. Conclusiones y trabajo futuro

A lo largo de este trabajo hemos comprobado que los algoritmos genéticos son una buena alternativa a las técnicas heurísticas para aproximar soluciones a problemas **NP** − hard. Se han seleccionado un grupo de configuraciones para dichos algoritmos que tienen un desempeño notable. Hemos comprobado también que, si bien algunos indicadores como la diversidad o porcentaje de factibilidad suelen tomarse como representativos de cuán bueno es un algoritmo genético, no son medidas rígidas y han de interpretarse con cautela. Este desarrollo nos ha permitido profundizar más en el funcionamiento de estos algoritmos y comprender mejor sus peculiaridades y limitaciones.

En cuanto a los problemas en particular que tratamos, hemos encontrado que no es posible determinar una configuración de parámetros del algoritmo que *resuelva* de modo coherente una instancia arbitraria de cualquier problema, es decir, la mejor configuración del algoritmo no es independiente de la estructura del problema.

De esta conclusión surge la cuestión de si las diferencias que se han expuesto son debidas a los problemas APO y PAPO (que estos sean más *difíciles* de tratar mediante algoritmos genéticos) o si son debidos a la naturaleza aleatoria de la programación genética. El teorema NFL nos proporciona un contexto para interpretar los resultados obtenidos, deduciendo que no es posible (en un marco completamente teórico) encontrar un algoritmo específico que rinda mejor en comparación con el resto en cualquier instancia de los problemas. Es necesario destacar que, aunque los algoritmos genéticos aquí expuestos no presenten la estabilidad que se desearía frente a la estructura del problema, hemos comprobado que superan considerablemente a las técnicas deterministas existentes, debido en parte a la incorporación de conocimiento previo sobre estos problemas al comportamiento del algoritmo. Aun así, los resultados son satisfactorios en comparación a las técnicas heurísticas.

En términos de un trabajo futuro, sería deseable profundizar en la relación encontrada para el problema APO donde se ha visto claramente que el rendimiento de los algoritmos genéticos empobrece a medida que la complejidad del problema crece. Este hecho, aquí tratado como hipótesis, requiere un estudio más detallado con el fin de determinar si este resultado es, en efecto, una relación existente o si bien es provocado por la gran aleatoriedad[8] que presentan dichos algoritmos.

[8]Con aleatoriedad nos referimos al hecho de que todos los operadores tanto genéticos como poblacionales requieren de un uso intensivo de variables aleatorias.

Bibliografía

[AWWB09] M. Affenzeller, S. Winkler, S. Wagner, and A. Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. Chapman & Hall/CRC, 1st edition, 2009.

[GH07] P. A. D. Gomez and D. F. Hougen. Initial population for genetic algorithms: A metric approach. In *Proceedings of the 2007 International Conference on Genetic and Evolutionary Methods, GEM 2007, June 25-28, 2007, Las Vegas, Nevada, USA*, pages 43–49, 2007.

[KSZ09] P. Konstantinos, N. Samaras, and D. Zissopoulos. Linear programming: Klee–minty examples. In *Encyclopedia of Optimization*, pages 1891–1897, Boston, MA, 2009. Springer US.

[LA11] G. Luque and E. Alba. Best practices in reporting results with parallel genetic algorithms. In *Parallel Genetic Algorithms: Theory and Real World Applications*, pages 31–51, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[LGT08] M. Lukasiewycz, M. Glaβ, and J. Teich. A feasibility-preserving crossover and mutation operator for constrained combinatorial problems. In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature — PPSN X - Volume 5199*, pages 919–928, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[LK08] M. Li and J. Kou. Crowding with nearest neighbors replacement for multiple species niching and building blocks preservation in binary multimodal functions optimization. *Journal of Heuristics*, 14(3):243–270, Jun 2008.

[RR02] C. R. Reeves and J. E. Rowe. *Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[RRR16] I. Rodríguez, F. Rubio, and P. Rabanal. Automatic media planning: Optimal advertisement placement problems. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 5170–5177, July 2016.

[SD07] S. N. Sivanandam and S. N. Deepa. *Introduction to Genetic Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2007.

[Shi12] O. M. Shir. Niching in evolutionary algorithms. In *Handbook of Natural Computing*, pages 1035–1069, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82, April 1997.

Lista de Algoritmos