Algoritmos Genéticos

Andrés Herrera Poyatos

8/11/2014

Contents

1 Introducción a los Problemas de Optimización **2**

1.1 Definición. Aplicación de la Inteligencia Artificial 2

1.2 Ejemplos de Problemas de Optimización 2

1.3 Heurísticas 4

2 Introducción a los Algoritmos Genéticos **5**

2.1 Operador de selección 5

2.2 Operador de mutación 5

2.3 Operador de cruce 7

2.4 Elitismo 8

2.5 Inicialización de la población 9

3 Modelos de Algoritmos Genéticos **10**

3.1 Algoritmo Genético Generacional 10

3.2 Algoritmo Genético Estacionario 10

3.3 Ejemplo: Aplicación al viajante de comercio 11

4 Comentarios finales **13**

4.1 ¿Por qué utilizar Algoritmos Genéticos? 13

4.2 Comentarios para el lector 13

1 Introducción a los Problemas de Optimización

1.1 Definición. Aplicación de la Inteligencia Artificial

Definición de inteligencia: Capacidad de entender o comprender.

La inteligencia es la pricipal característica diferenciadora entre los seres humanos y los demás animales. Entendemos, comprendemos y actuamos en consecuencia. La aplicación práctica de la inteligencia nos permite tomar mejores decisiones que el resto de especies.

Si indagamos un poco en la etimología de la propia palabra encontramos en su origen latino ***inteligere***, compuesta de *intus* (entre) y *legere* (escoger). Por lo que podemos deducir que ser inteligente es saber elegir la mejor opción entre las que se nos brinda para resolver un problema.

La resolución de un problema puede dividirse en dos partes:

- Encontrar alguna solución o todas las soluciones posibles al problema (**Conjunto de soluciones**).
- Elegir dentro del conjunto de soluciones encontrado la mejor solución posible.

Así pues, todo problema tiene un aspecto de optimización dado por el segundo paso.

Supongamos que se han encontrado todas las posibles soluciones a un problema dado. Denotamos S al conjunto de todas las soluciones. Para comparar dos soluciones y saber cual es mejor necesitamos una medida de la bondad de una solución. Así pues, supongamos que se dispone de una aplicación $f : S \to \mathbb{R}$ que nos proporciona esta medida. A esta función la denominaremos **función objetivo** del problema y a *f(s)* **fitness** de $s \in \mathbb{S}$. Definimos como **Problema de Optimización** a la búsqueda de la mejor solución del par (S, f). Distinguimos dos casos:

- **Problema de maximización**: Sean $s, s' \in S$ dos soluciones al problema. Se dice de maximización si verifica que s es mejor que s' si, y solo si, $f(s) > f(s')$.
- **Problema de minimización**: Sean $s, s' \in S$ dos soluciones al problema. Un problema de minimización es aquel en el que s es mejor que s' si, y solo si, $f(s) < f(s')$.

En definitiva, se trata de minimizar o maximizar f, hecho estudiado ampliamente para funciones en el análisis matemático. Sin embargo, f puede estar definida sobre cualquier conjunto y carecer de propiedades matemáticas que nos aporten una vía directa para la obtención de sus máximos o mínimos. Necesitamos otros métodos para conseguir nuestro objetivo.

Existen problemas donde el conjunto de soluciones posibles es muy grande. En estos problemas nuestra capacidad para evaluar las posibles soluciones y elegir la mejor se queda "pequeña". Incluso es factible encontrar problemas en los que la aplicación de algoritmos que recorran de forma óptima todo el conjunto de soluciones no sean viables en cuestión de tiempo. Tenemos que recurrir a otra herramienta: la **inteligencia artificial**.

Inteligencia Artificial: Desarrollo y utilización de ordenadores con los que se intenta reproducir los procesos de la inteligencia humana.

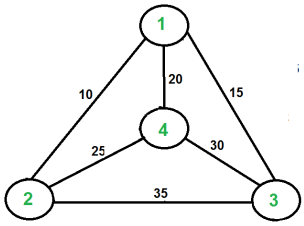
La aplicación de la Inteligencia Artificial a la resolución de problemas de optimización es la siguiente: proporcionar un algoritmo eficiente (las **heurísticas**) que busque en el conjunto de las soluciones hasta dar con una buena solución al problema (aunque no sea la mejor). En particular, **los Algoritmos Genéticos** permitirán obtener soluciones de problemas de optimización con un gran resultado.

1.2 Ejemplos de Problemas de Optimización

Se presentan a continuación 4 problemas de optimización que utilizaremos como ejemplo o propuestas de ejercicios a lo largo del texto.

1.2.1 El problema del viajante de comercio

Es ampliamente conocido por su nombre en inglés (**Travelling Salesman Problem**) y la abreviación del mismo (**TSP**), siendo uno de los problemas de optimización más estudiados. La formulación inicial del problema es la siguiente: dada una lista de ciudades con sus correspondientes coordenadas en el plano encontrar el ciclo más corto que pase por todas las ciudades una única vez. Un ejemplo sería el siguiente:



De forma más general y matemática, el problema del viajante de comercio consiste en, dado un grafo completo, *G(V,E)* con n vértices, y ponderado ($\omega : E \to \mathbb{R}^+$ nos proporciona la ponderación de cada arco), obtener aquel ciclo Hamiltoniano que minimice la suma de las ponderaciones de los arcos que lo constituyen. A tal suma se la llama coste de la solución. Así pues, la función objetivo es aquella que lleva cada solución a su coste correspondiente:

$$f : S \to \mathbb{R},\ f(s) = \sum_{\epsilon \in s} \omega(\epsilon)\ \forall s \in S$$

Una solución al viajante de comercio puede verse como el conjunto de arcos del correspondiente ciclo Hamiltoniano (usado en la definición anterior). Sin embargo, la representación más utilizada (y la que se usará a lo largo del texto) consiste en ver una solución como una permutación de los n primeros números naturales, donde cada número se identifica con un vértice del grafo y el orden de la permutación indica, dado un vértice, cuál es el siguiente a visitar. En la imagen anterior la mejor solución respondería a **(1,2,4,3)**. El número de soluciones posibles desde el punto de vista de esta representación es $n!$ (número de permutaciones de n elementos). Es claro que para un n pequeño el problema es inabordable desde el punto de vista de encontrar la mejor solución por el enorme número de soluciones posibles. Debemos pues abandonar la idea de encontrar la mejor solución con el fin de dar con una que al menos sea factible.

1.2.2 El problema de la máxima diversidad

Es conocido como *Maximum Diversity Problem*. Sea un conjunto finito $C = \{a_1, ..., a_n\}$, una medida de distancia sobre el mismo, $d : C \times C \to \mathbb{R}$ y $m \in \mathbb{N}$ verificando $m \leq n$. El problema de la máxima diversidad asociado a (C, d, m) consiste en encontrar un subconjunto de m elementos de C que maximice la suma de las distancias entre sus elementos. Por tanto, la función objetivo responde a:

$$f : S \to \mathbb{R},\ f(s = \{b_1, ..., b_m\}) = \sum_{i=1}^{m} \sum_{j=i+1}^{m} d(b_i, b_j)\ \forall s \in S$$

Este problema tiene una aplicación directa para encontrar subconjuntos de un tamaño dado en los que sus elementos sean lo más diferentes posible entre sí. Por ejemplo, en el diseño de productos que sean utilizados por consumidores de diferentes características.

Nótese que podemos representar una solución al problema como una cadena de ceros y unos donde un uno en la posición i-ésima indica que el elemento i-ésimo de C está en la solución y un cero indica lo contrario. Por tanto, todas las posibles soluciones serán vectores de $\mathbb{Z}_2^n$ verificando que la suma de sus componentes es m (tienen m unos).

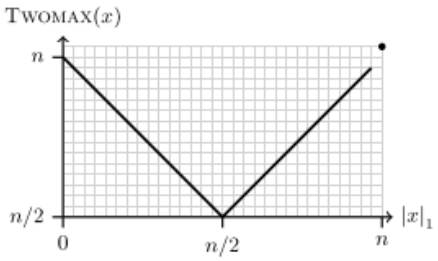
1.2.3 Binary Coding: Problemas de Optimización en Codificación Binaria

Son aquellos problemas de optimización cuya resolución se traduce en la optimización de una función del tipo: $f : \mathbb{Z}_2^n \to \mathbb{R}$. Las soluciones son codificadas mediante vectores de ceros y unos. Es claro que existen 2^n soluciones posibles al problema en cuestión.

Un ejemplo consiste en la optimización de la siguiente función denominadada $TWOMAX$[1]:

$$T : \mathbb{Z}_2^n \to \mathbb{R} \text{ dada por } T(x) = max\{|x|_0, |x|_1\} + \prod_{i=1}^{n} x_i \ \forall x \in \mathbb{Z}_2^n$$

donde $|x|_0$ responde al número de ceros de x y $|x|_1$ al número de unos. Gráficamente se tiene:



Es claro que la función tiene un máximo cuando todas las componentes del vector son unos. Es utilizada como test para algoritmos en este problema que aspiran a resolver funciones más complicadas.

Los problemas de codificación binaria fueron aquellos en los que inicialmente se formularon los algoritmos genéticos.

1.2.4 Real Coding: Optimización en variable real.

Consiste en optimizar una función del tipo: $f : \mathbb{R}^n \to \mathbb{R}$ continua. Es bien conocido que los computadores solo manejan un subconjunto finito de números reales. Por tanto, este problema a simple vista parece inabordable pues no se tienen todos los candidatos posibles. La continuidad de la función f es lo que nos permite intentar su resolución mediante computación pues nos garantiza que podemos acotar el error cometido en un elemento no representado por la computadora acercándonos lo suficiente al mismo.

1.3 Heurísticas

Dado determinado problema de optimización, una **heurística** es un proceso que calcula una solución que satisface determinados criterios de calidad con una eficiencia aceptable. Puede no calcular la mejor solución posible pero ser suficiente para nuestros objetivos. Las heurísticas son utilizadas esencialmente en **problemas NP** donde no es posible encontrar la mejor solución en un tiempo de computación reducido. Este es el caso de los problemas presentados anteriormente.

Las heurísticas suelen tener una complejidad algorítmica reducida e incluso ser procesos que dependen de iteraciones y podemos parar cuando consideremos oportuno.

Es habitual escuchar el término **metaheurística**, pudiendo causar confusión. Una metaheurística es una heurística de propósito general, basada en procedimientos genéricos y abstractos y que puede adaptarse al problema que se desee. Así pues, existen heurísticas específicas de determinados problemas que no pueden llamarse metaheurísticas pues carecen de la generalidad pertinente.

[1] D. E. Goldberg, C. Van Hoyweghen, and B. Naudts. *From twomax to the Ising model: Easy and hard symmetrical problems.* In Proc. of GECCO '02, pages 626–633. Morgan Kaufmann, 2002

2 Introducción a los Algoritmos Genéticos

Los **algoritmos genéticos** son metaheurísticas basadas en poblaciones y en los conceptos de evolución y genética. En la naturaleza, las poblaciones de individuos evolucionan con el tiempo gracias a tres factores: la **selección natural**, la **reproducción** y la **mutación**. El primero de ellos consiste en la presión ejercida por parte del entorno en la población. Este hecho provoca la supervivencia de solo un subconjunto de la población, aquellos individuos que han demostrado ser más fuertes. La reproducción da lugar a nuevas generaciones de individuos en algunos de los cuales se ha producido una mutación, introducciendo variedad en la población. La idea de los algoritmos genéticos es imitar este proceso obteniendo así mejoras progresivas en un conjunto de soluciones denominado **población** del algoritmo genético. Formalizamos el algoritmo a continuación.

Consideremos a partir de este momento un problema de otimización con conjunto de soluciones S y función objetivo *f*. Sea, además, un conjunto de m soluciones, denotado *P(0)*, que conforma la población inicial del algoritmo genético. A la población en un tiempo $t \in \mathbb{N}$ se la denota *P(t)*. Definimos un algoritmo genético como una heurística probabilística que genera a partir de la población *P(t)* la siguiente población *P(t+1)* utilizando operadores basados en la genética y evolución: el operador de selección, operador de cruce y operador de mutación.

Nos proponemos a desarrollar a continuación los elementos que conforman el algoritmo genético:

2.1 Operador de selección

Un **operador de selección**[2] es un proceso que permite seleccionar soluciones de una población de forma aleatoria pero manteniendo cierto criterio de calidad. Debe imitar a la idea de selección natural. De esta forma, las mejores soluciones "sobrevivirán" al operador de selección siendo escogidas para generar hijos mediante el operador de cruce. Así pues, el operador de selección puede dar lugar a una mayor o menor **presión selectiva** que determinará el comportamiento del algoritmo.

Un operador de selección suele ser realizado con reemplazamiento. En una minoría de casos la selección es sin reemplazamiento, teniendo menos margen de maniobra. A continuación se muestran diferentes operadores de selección:

- **Selección por Torneo**:
  Es la selección más utilizada en los algoritmos genéticos. Consiste en seleccionar de forma aleatoria k ($1 \leq k \leq m$) soluciones y tomar la mejor entre las mismas. Nótese que con esta selección las peores soluciones de la población son muy penalizadas pues las *k-1* peores soluciones nunca serán elegidas para la operación de cruce. Para no realizar una excesiva presión se suele tomar $k = 2$.
- **Selección por Ruleta:**
  Consiste en asignar a cada solución s una probabilidad de elección. Esta probabilidad será proporcional a *f(s)* en el caso de un problema de maximización $\left(p(s) = \frac{f(s)}{\sum_{s' \in P(t)} f(s')}\right)$ o inversamente proporcional en el caso de un problema de minimización $\left(p(s) = \frac{1/f(s)}{\sum_{s' \in P(t)} 1/f(s')}\right)$. Es el mecanismo de selección clásico.

2.2 Operador de mutación

Un **operador de mutación** consiste en un mecanismo que, dada una solución $s \in S$, proporcione según determinada ley de probabilidad una solución que sea "similar" a la anterior. El concepto de similaridad es muy relativo. Le damos a continuación cierto rigor al mismo con el concepto de vecinos.

[2] Comparativa entre diferentes mecanismos de selección:
Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. G. J. E. Rawlins (Ed.), Foundations of Genetic Algorithms, p. 69-93

Supongamos que se dispone de una aplicación $v : S \to \mathcal{P}(S)$ tal que, dada $s \in S$ nos proporciona un subconjunto de S que contiene a s. A v se la denomina **función de vecindad**. Así pues, una solución *s'* es **vecina** de s, en términos de v, si pertenece a *v(s)*. Podemos ver pues un operador de mutación como un proceso que dada una función de vecindad y $s \in S$ proporciona, bajo determinada ley de probabilidad, un vecino de S.

2.2.1 Ejemplos

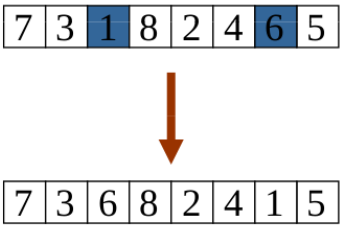
A continuación se muestran ejemplos de operadores de mutación en diversos problemas de optimización:

2.2.1.1 Problema del viajante de comercio

- **Mutación por Intercambio**:
  En este caso la función de vecindad responde a:

  $$v(s) = \{\sigma(s) : \sigma \text{ es una transposición de los n primeros números naturales}\}$$

  Además, se toma de forma equiprobable un elemento de *v(s)* como mutación de *s*. En definitiva, consiste en intercambiar dos ciudades de posición en el recorrido.

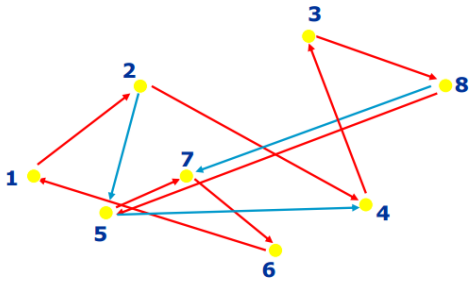


- **Mutación por Inserción**: En este caso la función de vecindad responde a:

  $$v((x(1), x(2), ..., x(n))) = \{(x(1), ..., x(i-1), x(j), x(i+1), ..., x(j-1), x(j+1), ..., x(n)) : i, j \in \{1, .., n\}\}$$

  Es decir, cualquier inserción de un elemento de la propia solución en otra posición de la misma. Se toma de forma equiprobable un elemento de *v(s)* como mutación de *s*. Un ejemplo es el siguiente:

  $$(\mathbf{1\ 2\ \_\ 4\ 3\ 8\ '5'\ 7\ 6}) \Rightarrow (\mathbf{1\ 2\ 5\ 4\ 3\ 8\ 7\ 6})$$



2.2.1.2 Problemas de Optimización en Codificación Binaria

Las mutaciones en estos problemas son sencillas, basta cambiar una o varias componentes de la solución así como intercambiar componentes de posición.

2.2.1.3 Problemas de Optimización en Variable Real

El operador de mutación usual consiste en sumar a una componente al azar del vector un valor pequeño. Cualquier otra mutación que siga esta filosofía es válida para el problema.

2.3 Operador de cruce

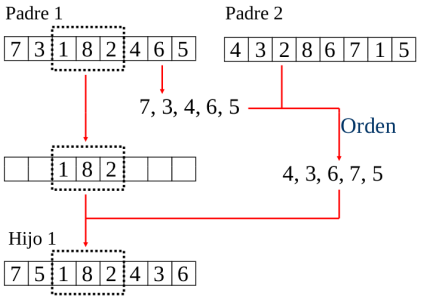
Un **operador de cruce** es un proceso que, dadas dos soluciones $s, s' \in S$, proporciona una nueva solución que sea "similar" a las dos anteriores. A esta nueva solución se la denomina **hijo** mientras que a las dos soluciones previas se las denomina **padres**. En definitiva, la idea es la misma que la del operador de mutación, una aplicación $v : S \times S \to \mathcal{P}(S)$ que para cada par de soluciones(s,s') proporciona un conjunto de soluciones $v(s, s')$ de las cuales se elige la solución hijo. En este caso es más dificultoso proporcionar en la práctica la aplicación v pues los operadores de cruce buscan que las soluciones obtenidas hereden las características de los padres en lugar de interpretabilidad.

2.3.1 Ejemplos

A continuación se muestran ejemplos de operadores de cruce en diversos problemas de optimización:

2.3.1.1 Problema del viajante de comercio [3]

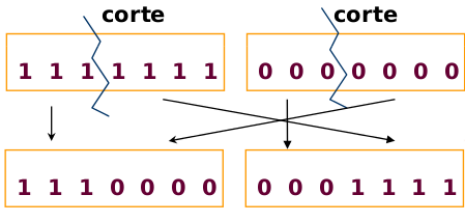
- **Operador de cruce OX**:
  En la representación dada, el orden de los vértices del grafo es muy importante pues es el que decide los arcos que componen la solución. Con esta filosofía, el operador de cruce pretende cruzar dos padres con especial atención al orden de sus representaciones. Consiste en tomar una subcadena de la representación de uno de los padres que pasa como tal al hijo. El resto de elementos de la solución hijo se toman respetando el orden de la representación del otro padre:



2.3.1.2 Problemas de optimización en Codificación Binaria

- **Cruce por un punto**:
  Sean dos soluciones $s = (s_1, .., s_n)$ y $s' = (s'_1, .., s'_n)$, se selecciona aleatoriamente $i \in 1, .., n$ y se toma como hijo el vector $(s_1, .., s_i, s'_{i+1}, ..., s'_n)$. En la siguiente imagen se muestra la obtención de dos hijos mediante este cruce:

[3] Una comparación entre múltiples operadores de cruce y de mutación para el problema del viajante de comercio: Larrañaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., & Dizdarevic, S. (1999). *Genetic algorithms for the travelling salesman problem: A review of representations and operators.* Artificial Intelligence Review, vol. 13, no 2, p. 129-170.



- **Cruce Uniforme**:
  Dadas dos soluciones, se genera un hijo tomando aleatoriamente varias componentes del primer padre y el resto del segundo padre. Nótese que las componentes que sean comunes a los dos padres mantienen el mismo valor en el hijo.

- **HUX**:
  Se define la **distancia de Hamming** entre dos vectores como el número de componentes en las que difieren. El uso de esta distancia es lógico en $\mathbb{Z}_2^n$. El cruce *HUX* consiste en la aplicación del Cruce Uniforme de forma que el hijo se encuentre a la misma la distancia de Hamming de ambos padres. Sean dos soluciones *s* y *s'* con distancia de Hamming entre ellas *m*, se construye un hijo de ambas mediante el cruce *HUX* tomando las *n - m* componentes comunes y, aleatoriamente, *m/2* componentes no comunes de un padre y el resto del otro.

2.3.1.3 Problemas de Optimización en Variable Real

Se pueden utilizar los tres operadores de cruce anteriores. Otra opción común es calcular un hijo como la media de los dos padres.

2.4 Elitismo

Podríamos con los elementos desarrollados definir un algoritmo genético que dada una población *P(t)* con *n* soluciones toma mediante determinado operador de selección *n* soluciones de la misma y las cruza (produciendo una mutación con baja probabilidad) para obtener *P(t+1)*.

Este algoritmo respeta completamente la evolución natural. Sin embargo, esta puede mejorarse. Si generamos una nueva población de forma completa mediante los operadores genéticos las soluciones que la conforman no tienen por qué ser mejores que las soluciones de la población anterior. Como consecuencia, en determinado punto de la ejecución del algoritmo si se padece de excesiva mala suerte la población puede empeorar de forma severa, no obteniendo buenos resultados a posteriori. Es conveniente, por tanto, exigir el cumplimiento de la siguiente desigualdad:

$$f(MS(t+1)) \text{ es mejor que } f(MS(t)) \ \forall t \in \mathbb{N}$$

donde *MS(t)* proporciona la mejor solución de *P(t)*. Es decir, la mejor solución de la población *t* debe ser mejor que las de sus predecesoras. Una forma de conseguir que el algoritmo verifique la desigualdad anterior consiste en aplicar el concepto de **elitismo**: mantener las mejores soluciones obtenidas en la población creada.

Una forma habitual de aplicar este concepto consiste en añadir directamente a *P(t+1)* la mejor solución de *P(t)*. Otra opción consiste en sustituir la peor solución de *P(t+1)* por la mejor de *P(t)* Existen otros conceptos que dan lugar a elitismo como la **competición entre padres e hijos** donde la idea principal es tomar en *P(t+1)* las mejores soluciones entre padres e hijos. Se profundizará en este concepto en nuevas extensiones del texto.

2.5 Inicialización de la población

La población inicial suele ser generada de forma aleatoria pues la obtención de una buena solución la realizan los mecanismos genéticos, no es necesario perder tiempo computacional en obtener una población inicial de calidad. Sin embargo, es válido obtener la población mediante otras heurísticas, como algoritmos voraces aleatorizados, con el requisito de que produzcan soluciones diferentes entre sí. El tamaño de la población oscila entre 50 y 100 individuos en la literatura especializada, existiendo algunos algoritmos que sí pueden funcionar con tamaños de población más bajos.

3 Modelos de Algoritmos Genéticos

Se presentan a continuación los modelos usuales de algoritmos genéticos:

3.1 Algoritmo Genético Generacional

Su funcionamiento se adivina fácilmente tras la discursión anterior. Consiste en generar de forma completa una nueva población *P(t+1)* a partir de *P(t)*. El proceso a seguir es el siguiente. Mediante determinada selección se toman *n/2* parejas de padres (donde *n* es el tamaño de *P(t)*). Habiendo fijado de antemano una probabilidad de cruce (*prob_cruce*), que suele oscilar en torno a 0.7 y 0.8 en la literatura especializada, se cruza una pareja con probabilidad *prob_cruce* dando a lugar a dos hijos sobre los que se aplica el operador de mutación independientemente con una probrabilidad dada por *prob_mutacion* que suele oscilar entre 0 y 0.2. En caso de no cruzarse la pareja de soluciones, se aplica la mutación con idéntica probabilidad a los padres. La pareja resultante de este proceso pasa a la *P(t+1)*. Se utiliza elitismo sustituyendo la peor solución de *P(t+1)* por la mejor de *P(t)*

Se presenta a continuación el código correspondiente:

```
iteraciónAlgoritmoGenéticoGeneracional()
    P(t+1) <- Vacío;
    while (size(P(t+1)) < size(P(t))) do
        padre1 <- seleccion(P(t));
        padre2 <- seleccion(P(t));
        if (realEntre0y1() < prob_cruce)
            hijo1 <- cruce(padre1, padre2);
            hijo2 <- cruce(padre2, padre1);
        else
            hijo1 <- padre1;
            hijo2 <- padre2;
        endif
        if (realEntre0y1() < prob_mutacion)
            mutacion(hijo1);
        endif
        if (realEntre0y1() < prob_mutacion)
            mutacion(hijo2);
        endif
        P(t+1) <- P(t+1) U {hijo1,hijo2};
    endwhile
    PS(t+1) <- MS(t);
    t++;
```

La aplicación de probabilidades para decidir si aplicar o no el operador de cruce pretende que el cambio de una generación a otra no sea tan drástico manteniendo parejas de soluciones no cruzadas. En el caso del operador de mutación la inclusión de esta probabilidad es obligada para añadir nuevo material genético a la población y evitar que esta tienda a asemejarse al mejor de la misma tras varias iteraciones del algoritmo. Sin embargo, esta probabilidad se mantiene en valores bajos pues es claro que la mutación dismminuye genéricamente la calidad de la solución sobre la que se aplica.

3.2 Algoritmo Genético Estacionario

Un algoritmo genético estacionario produce cambios ligeros y progresivos en la población en cada uno de los cuales se mantiene el núcleo de la misma. Consiste en la selección de una pareja de soluciones mediante

determinado operador de selección. Esta pareja se cruza obteniendo dos hijos sobre los que se aplica el operador de mutación con determinada probabilidad. Los dos hijos se introducen en la población mediante determinada estrategia de reemplazamiento. Varias posibles son las siguientes:

- **Reemplazar al peor de la población (RW):**
  Se reemplaza la peor solución de la población. Genera una alta presión selectiva y una rápida convergencia (Las soluciones que conforman la población rápidamente tienden a ser muy similares entre sí).
- **Torneo Restringido (RTS):**
  Se eligen k soluciones de la población de forma aleatoria (normalmente $k = 3$). De entre estas k soluciones se reemplaza aquella más parecida a la nueva solución. De esta forma la población mantiene soluciones diferentes entre sí.
- **Peor entre semejantes (WAMS):**
  Se reemplaza la peor solución del conjunto de las k (normalmente $k = 3$) soluciones más parecidas a la nueva solución. Busca equilibrio entre diversidad y presión selectiva a costa de una búsqueda lineal en la población.
- **Algoritmo de Crowding Determinístico (DC):**
  El hijo reemplaza a su padre más parecido. Mantiene la diversidad de la población.

Se presenta a continuación el código correspondiente al algoritmo genético estacionario con reemplazamiento al peor de la población:

```
iteraciónAlgoritmoGenéticoEstacionario()
    padre1 <- seleccion(P(t));
    padre2 <- seleccion(P(t));
    hijo1 <- cruce(padre1, padre2);
    hijo2 <- cruce(padre2, padre1);
    if (realEntre0y1() < prob_mutacion)
        mutacion(hijo1);
    endif
    if (realEntre0y1() < prob_mutacion)
        mutacion(hijo2);
    endif
    PS(t) <- hijo1;
    PS(t) <- hijo2;
    P(t+1) <- P(t);
    t++;
```

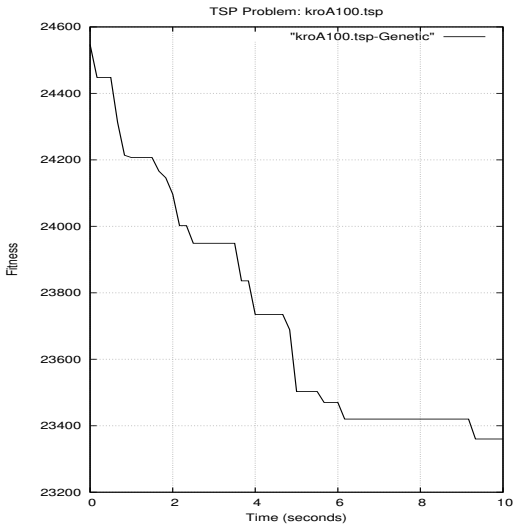
La estrategia de reemplazamiento elegida depende del problema en cuestión y del comportamiento que deseemos que mantenga el algoritmo. Debe mantener la sinergia con el resto de operadores genéticos elegidos. Nótese que para la elección de una estrategia que sustituya aquellas soluciones similares etre sí es necesario tener una medida de similitud entre soluciones. Cada problema puede presentar diferentes medidas de distancia para soluciones del mismo. Un ejemplo es la distancia de Hamming introducida a lo largo del texto.

Actualmente el uso de los algoritmos genéticos estacionarios ha disminuido por la aparición de modelos generacionales con muy buenos resultados, sin embargo, siguen siendo un pilar básico de la literatura especializada.

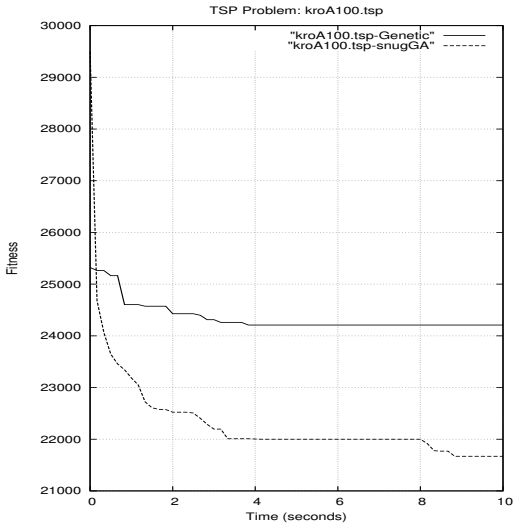
3.3 Ejemplo: Aplicación al viajante de comercio

La siguiente imagen muestra la evolución de una ejecución del algoritmo genético generacional en el problema **kroA.tsp** de la librería TSPLIB[4].

[4]Reinelt, G. (1991). *TSPLIB—A traveling salesman problem library.* ORSA Journal on Computing, vol 3, no 4, p. 376- 384. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html



Se puede observar la mejora progresiva de la mejor solución de la población en función del tiempo. El algoritmo utilizado consta del operador de cruce OX, la mutación por intercambio y la selección por torneo. Si mejorásemos la sinergia entre los operadores del algoritmo genético los resultados pueden ser muy buenos. En la siguiente imagen se comparan el mismo algoritmo genético y un algoritmo genético desarrollado por mi (algoritmo genético con diversificación voraz de la población y competición entre padres e hijos):



4 Comentarios finales

4.1 ¿Por qué utilizar Algoritmos Genéticos?

Los algoritmos genéticos son empleados en problemas de optimización en ingeniería con asiduidad. Un ejemplo de estos problemas es la obtención de componentes aerodinámicas para coches o aviones. Podemos argumentar una serie de pros y contras en relación a estos algoritmos:

PROS:

- Bajo coste computacional.
- Muy customizables.
- Si la función objetivo del problema varía el algoritmo mantiene su funcionamiento de forma intacta partiendo de las mismas soluciones obtenidas anteriormente.
- Proporciona una población de soluciones como resultado final del algoritmo, teniendo más donde elegir.
- Son válidos para optimización multimodal en la que se pretende conseguir todos los máximos o mínimos atendiendo a la función objetivo del problema. Para ello se utilizan con otras variaciones como nichos.
- Buenos resultados si se elige modelo adecuado.

CONTRAS:

- Poca fundamentación matemática.
- Un algoritmo genético que no haya sido adaptado al problema o mal planteado en cuanto a sinergia entre los operadores puede dar muy mal resultado.

4.2 Comentarios para el lector

Esta es una explicación básica sobre los algoritmos genéticos. Existe mucho material al respecto en la literatura especializada y sigue siendo una temática de investigación activa. Para una mayor profundización en la temática sentiros libres de escribirme a mi dirección de correo: andreshp9@gmail.com

Espero poder preparar otra sesión con otros algoritmos genéticos como CHC, Micro-GA, meméticos... en la que se haga incapié en la importancia de mantener una población con diversidad y al mismo tiempo una gran presión selectiva.

Se deja como ejercicio el desarrollo de operadores para el problema de la máxima diversidad, hecho sencillo tras la base dada.