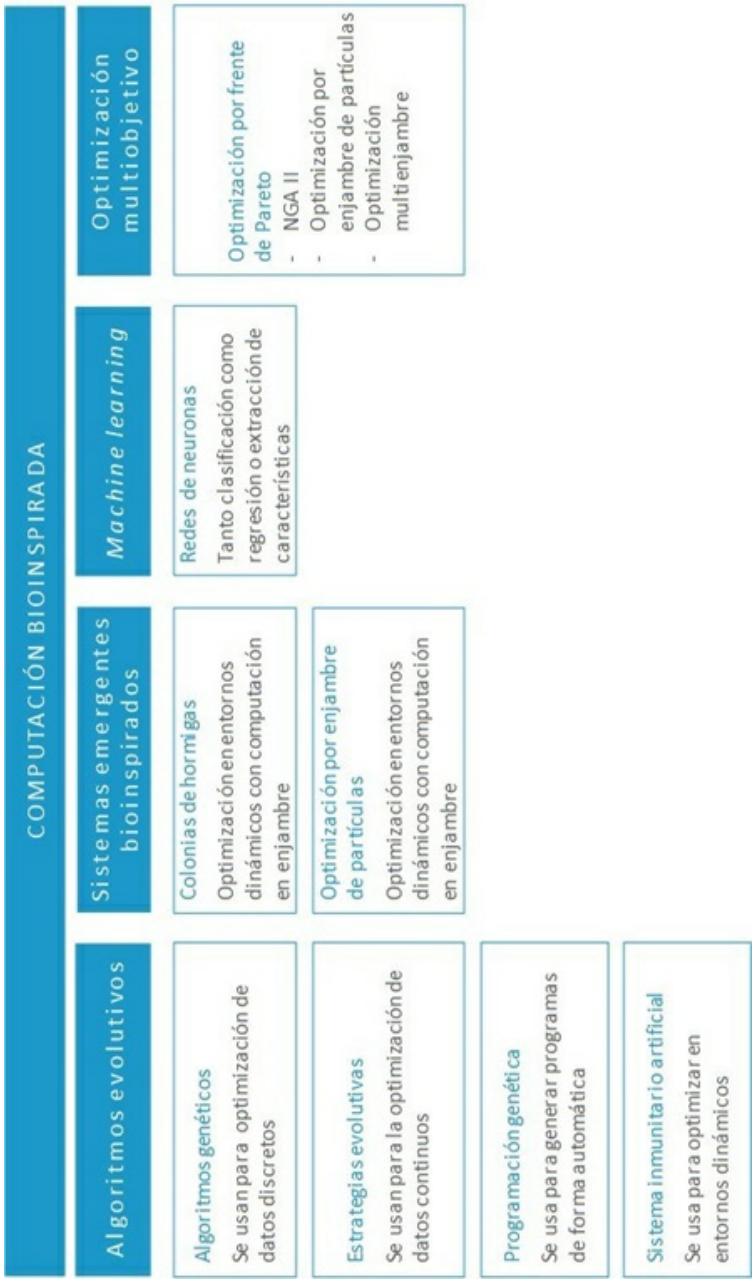
Investigación y Gestión de Proyectos en Inteligencia Artificial

Tema 11. Investigación en computación bioinspirada

Índice

Esquema



11.1. Introducción y objetivos

En este tema se explica a alto nivel las **diferentes técnicas y algoritmos de inteligencia artificial que han surgido con inspiración en la biología.** Así pues, en este tema presentaremos un conjunto de técnicas y algoritmos o aproximaciones que se han inspirado de alguna forma en sistemas presentes en la naturaleza. Este campo es inmenso y solo queremos dar unas pinceladas sobre el tema, para que tengas un concepto general sobre este tipo de tecnologías.

Tradicionalmente, los humanos hemos intentado imitar el comportamiento de ciertos individuos o procesos naturales para el avance tecnológico. No siempre se han adaptado los sistemas biológicos tal cual existen en la naturaleza (por ejemplo, el hombre no vuela como las aves), pero su estudio siempre ha sido un campo de inspiración para el avance científico y técnico.

No hay que confundir la **computación biológica con la biología computacional.** La computación biológica estudia cómo podemos utilizar elementos de naturaleza biológica para obtener mejores resultados a problemas en la computación. Sin embargo, la biología computacional es el uso de técnicas de computación en el estudio de la biología molecular, procesado de ADN, etc. Las ideas detrás de ambos enfoques son similares pero la finalidad no tiene nada que ver.

De forma detallada, los **objetivos** que persigue esta unidad son:

- Ser capaz de entender cómo funcionan estos sistemas bioinspirados.
- Conocer los tipos de algoritmos y problemas que existen y que pueden resolverse con ellos.
- Identificar las principales técnicas y algoritmos, en especial, los relacionados con la computación evolutiva.

Ideas clave

- Conocer los proyectos que se pueden crear o resolver usando computación bioinspirada.

11.2. Introducción a la computación bioinspirada

Como hemos definido anteriormente, la computación bioinspirada o computación biológica es un **área de la inteligencia artificial que se apoya en soluciones inspiradas en la biología para resolver problemas computacionales.** La variedad de algoritmos y de técnicas es enorme y, además, muchas de ellas están siendo la punta de lanza de la IA (inteligencia artificial) actualmente, obteniendo unos resultados asombrosos. Por la inmensidad del área a abarcar, vamos a centrarnos en los **algoritmos de computación evolutiva** y después mencionaremos otros enfoques con menor profundidad.

Para comprender la computación evolutiva tenemos que acudir a las bases de la **teoría de la evolución propuesta por Charles Darwin** (1809-1882) y su **teoría de la selección natural.**

Esta teoría surge después de visitar las islas Galápagos en una expedición geológica. En ellas Darwin observó algunas anécdotas biológicas muy interesantes que le hizo interesarse por la evolución de las especies, ya que encontró similitudes con especies de América pero a la vez diferencias. Darwin en realidad no era biólogo sino geólogo, y no fue hasta este momento cuando comenzó a interesarse más a fondo en temas de biología y de especies.

Por ejemplo, se percató de que, en cada isla, un grupo de especies diferentes de pájaros, muy similares en color, aspecto y tamaño, tenían modificaciones en su pico adaptadas a las necesidades de obtención de alimento, que eran diferentes en cada isla. Por ejemplo, unos tenían el pico más largo, o más curvo o de diferente tamaño.

Tardo varios años en recopilar toda la información que necesitaba para formular su teoría. En ella, describe que las especies evolucionan unas de otras, ramificándose a partir de padres comunes. Algunas de estas especies terminan en extinción y otras

volviéndose a ramificar en nuevas especies, lo que le confiere una forma de árbol. También relacionó el desarrollo embrionario de un individuo con el desarrollo evolutivo de sus estirpes (esto lo extrajo de los fósiles que recolectó en su vida como geólogo), y que las islas u otras barreras geológicas hacían que las especies se adaptaran al entorno de esas zonas.

Darwin sabía que las especies evolucionaban y que el medio en el que se desarrollaban influía, pero no sabía el mecanismo de cómo evolucionaban estas especies. Esto se descubrió paralelamente por **Mendel** (1957-1968) que describió cómo funcionaba la mutación haciendo unos experimentos con plantas de guisantes.

De esta forma, años después con el descubrimiento del ADN, se conformó la teoría completa que sustenta lo que hoy conocemos como la **teoría de la evolución.**

La teoría de la evolución dice que la presión ambiental (reproductiva, por obtener alimentación, competencia con otros animales) es el vehículo del cambio en las especies y el mecanismo del cambio es el cruce genético y la mutación. Es decir, el cruce entre dos especies provoca que parte del ADN de una de ellas se mezcle con parte del ADN de la otra. De forma que el nuevo individuo no tiene exactamente el mismo ADN que los progenitores, sino una mezcla de ambos. Además de esto, otro mecanismo para el cambio es la mutación, que consiste en pequeños cambios en los genes que se producen o no, de forma aleatoria en cada réplica del ADN. De esta forma, un nuevo individuo puede tener un ADN diferente al de sus progenitores. Algunos de estos cambios producen individuos más aptos para el medio en el que se desenvuelven. Mientras que otros cambios hacen a estos individuos menos aptos para su supervivencia. La presión selectiva hace que no todos los individuos acaben transmitiendo sus genes a sus descendientes, bien porque no consiguen aparearse (no son el macho dominante con ese derecho en una manada, por ejemplo) o bien porque mueren antes de poder hacerlo. De esta forma, la probabilidad de que un cambio genético que perjudique la supervivencia de un individuo en un entorno concreto se transmita a los descendientes es menor que la probabilidad de que lo

haga uno beneficioso, ya que este tendrá más probabilidades de reproducirse. Y el tiempo hace el resto. De forma que, en sucesivas generaciones, los individuos con modificaciones que le permiten adaptarse mejor al medio han proliferado y han transmitido estas modificaciones a la mayoría de la población.

Si miramos esto desde un punto de vista computacional, lo que hace la selección natural es un **método de optimización.** La selección natural optimiza a los individuos para resolver una tarea concreta. Que en el caso de la biología es sobrevivir y reproducirse, con todo lo que ello implica. Si una especie no se adapta a los cambios desaparece, lo que da lugar a las extinciones, ya sea porque no se adapten a un cambio repentino del medio, ya sea porque una versión mejorada de ellos mismos los sustituya en su nicho ecológico.

Ahora extrapolemos esto a un problema de computación. ¿Podemos generar una solución a un problema de computación y hacerla evolucionar para que encuentre la forma de resolver ese problema de la forma más óptima posible?

De esta pregunta surge la **computación evolutiva.**

Computación evolutiva

La computación evolutiva es una **rama de la inteligencia artificial que busca resolver problemas de optimización, inspirándose en los mecanismos de evolución biológica.** Existen diferentes métodos y algoritmos que se pueden englobar dentro de este campo como los algoritmos genéticos, las estrategias evolutivas, la programación genética etc. Ya los desgranaremos en el resto del tema. Pero por ahora vamos a asentar las bases de los conceptos en los que se sustentan todas ellas.

Podemos definir **gen como la unidad mínima de transferencia genética.** Es decir, la unidad mínima que es capaz de sintetizar una macromolécula con función celular. Estos genes tienen memoria y esto permite la herencia ente descendientes.

Ideas clave

El **genotipo** es el conjunto de todos los genes de un individuo.

El **fenotipo** es la expresión del genotipo en un individuo concreto, dentro de un ambiente concreto. Es decir, la implementación de ese genotipo.

Ejemplo: hay un gen que codifica el color de los ojos de un individuo. Su genotipo sería la secuencia de bases del gen en concreto y su fenotipo el color que genera al final.

Con estas definiciones, lo que buscan los algoritmos evolutivos es crear una analogía con la evolución natural. Así pues, habrá que representar una cadena de ADN con información mínima útil para realizar una tarea (gen). Estos genes se reproducirán (o no) cruzándose con otros genes y ocasionalmente se cambiará la información de alguno de estos genes mediante una mutación. Este proceso se repetirá hasta encontrar un individuo lo suficientemente bueno como para que resuelva la tarea que debe realizar.

¿Y cómo sabemos si resuelve la tarea? Pues ejecutando dicha tarea y «puntuando» lo bien que lo ha hecho. Esa puntuación es una valoración de lo adaptado que está el individuo al medio.

Aplicaciones

Las aplicaciones de este tipo de algoritmos evolutivos son muchas y está enmarcadas dentro de los algoritmos de optimización. Son capaces de encontrar una forma eficiente de realizar una tarea. Vistos así puede el lector pensar que no son muy diferentes a un algoritmo de búsqueda, que busca la mejor solución entre las posibles soluciones y es cierto. La mayoría de estos algoritmos pueden ser considerados algoritmos de búsqueda, pero tiene ciertas peculiaridades que los hacen especialmente efectivos en determinados entornos. Y es que son bastante buenos en encontrar soluciones subóptimas en problemas que tienen una alta complejidad y donde algoritmos como A* no son viables. También son bastante

robustos a mínimos locales en las búsquedas debido a su componente aleatorio. Por otro lado, su complejidad computacional viene determinada principalmente por **lo costoso que sea evaluar la calidad de la solución.** Es decir, la calidad que cada individuo tiene para resolver el problema. Sí es muy costoso hacerlo, estos algoritmos suelen ser demasiado costosos ya que normalmente necesitan poblaciones grandes de individuos (o soluciones), y habrá que ejecutar la función de evaluación para cada individuo y para cada generación.

Más en concreto, se han utilizado en **diferentes campos** como:

- La robótica.
- La optimización de procesos.
- Fabricación industrial.
- Economía.
- Aprendizaje automático en por ejemplo la optimización de algoritmos *de machine learning* como redes de neuronales.

A continuación, veremos algunas de estos métodos con más detalle.

11.3. Algoritmos bioinspirados

A continuación, explicaremos las principales técnicas englobadas en los denominados algoritmos bioinspirados. Haremos más hincapié en los más comunes y utilizados, basándonos en las explicaciones de la selección natural que hemos realizado en el anterior apartado.

Algoritmos genéticos

Los algoritmos genéticos son unos **algoritmos evolutivos que se basan en los principios explicados de la selección natural.** Estos algoritmos pretenden resolver problemas de optimización y búsqueda representando la solución en forma de genes. Estos se cruzan y mutan en sucesivos ciclos para conseguir individuos que son capaces de resolver el problema cada vez mejor.

Así pues, debemos obtener la mejor solución dentro del espacio de soluciones que resuelva un problema planteado, atendiendo a un criterio de evaluación del rendimiento de la solución obtenida. Esto se puede aplicar por tanto a:

- Búsqueda en un espacio de estados.
- Clasificación: cuál es el procedimiento que mejor clasifica.
- Predicción: encontrar el procedimiento que tenga menos fallos.
- Control: procedimiento que maximice una tarea de control.

En los algoritmos genéticos es muy importante la **codificación.** Es la principal tarea del diseñador del algoritmo. Los algoritmos genéticos deben codificar las diferentes soluciones como un vector o matriz de genes. Estos vectores de genes se los

denomina individuos y codifican una solución particular dentro del espacio de soluciones.

Una **población** es el conjunto de soluciones (individuos) que disponemos en un momento dado. Inicialmente esta población se puede generar de forma aleatoria o con algo más de conocimiento del dominio. A cada uno de estos individuos se le aplica una serie de operadores genéticos que transforman una población en otra.

¿Y cuáles son esos operadores genéticos?

- **Selección.** Se selecciona a los mejores candidatos para la reproducción. Se genera una población intermedia del mismo tamaño que la original normalmente. Hay diferentes técnicas a la hora de seleccionar cuáles son los individuos candidatos para reproducirse.
- **Cruce.** Se seleccionan de dos en dos los individuos de esa población intermedia y se cruzan de alguna forma. Hay diferentes técnicas de cruce como cruce simple por la mitad, cruce uniforme donde cada gen se cruza con su homónimo en la pareja, cruzar por múltiples puntos, etc.
- **Mutación.** Se modifica de forma aleatoria mediante una probabilidad de mutación los genes de los individuos recién cruzados.
- **Inversión.** Se invierten algunos de los genes (en caso de que la codificación sea binaria), también se puede coger un grupo de ellos y darles la vuelta.

Para finalizar tenemos una función que se denomina **función de *fitness*** que evalúa las soluciones (los individuos) para determinar cuáles son los más aptos, es decir, los que mejor resuelven el problema. Esta función de fitness debe resumir el comportamiento del individuo con un número para poder comparar unos con otros.

Ideas clave

Los algoritmos genéticos son muy eficientes a la hora de encontrar soluciones más o menos optimas a un problema. Y son propensos a una convergencia prematura en un mínimo local. Sobre todo, si solo usamos cruce. Esto es debido a **que ciertos rasgos genéticos se pierden** (a esto se le denomina perdida de alelos) en los cruces con lo que se pierde variabilidad genética. La mutación aquí es un factor importante para mantener esa variabilidad genética. Después de unas cuantas generaciones, todos los individuos tienen un valor de adecuación o fitness similar, por lo tanto, deja de haber **presión selectiva** y, por tanto, se deja de evolucionar (todos tiene la misma probabilidad de reproducirse). Par evitarlo, **surgen diferentes técnicas** que favorecen que aumente o disminuya la presión selectiva para dar oportunidades a que rasgos óptimos en individuos no óptimos puedan formar parte de soluciones mejores y no se pierdan por el camino. Para ellos hay diferentes tipos de selección.

Tipos de selección

La selección por ruleta es similar a tirar un dado donde los individuos con mejor fitness tienen más probabilidades de acertar. Acertar en este caso es reproducirse.

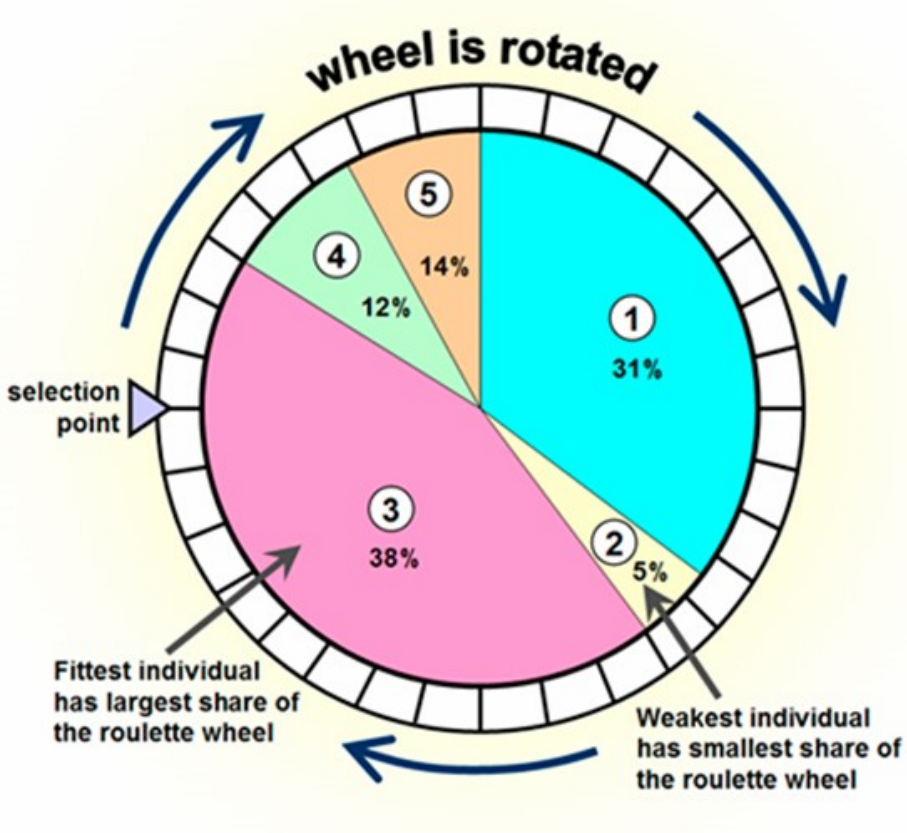


Figura 1. Selección por ruleta. Fuente: https://stackoverflow.com/questions/23183862/genetic-programming-difference-between-roulette-rank-and-tournament-selection

En la **selección jerárquica** se ordena a la población de mayor a menor según su fitness y se selecciona el individuo en proporción al rango que ocupa calculando una probabilidad.

En la selección por **torneos** se eligen K elementos de la población y se selecciona aquel que tenga mejor *fitness*. Normalmente este tipo de selección relaja la presión selectiva lo que hace que la diversidad genética sea mayor.

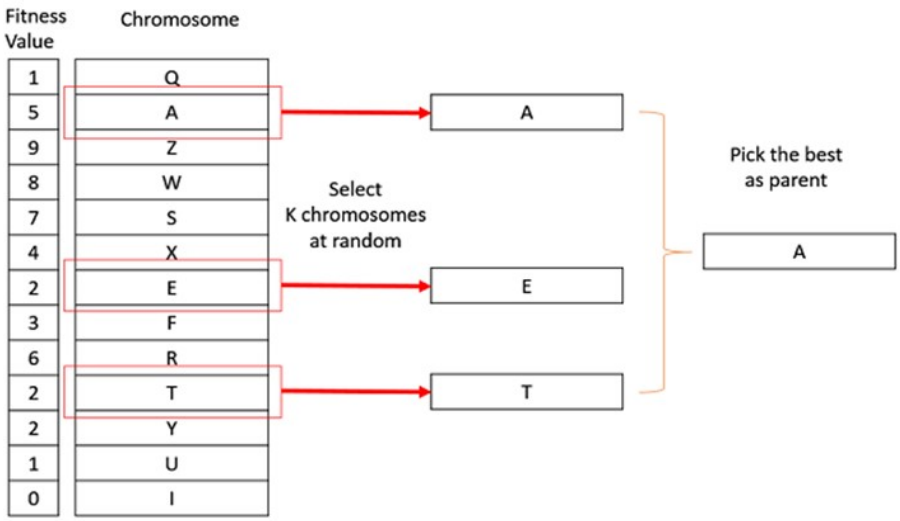


Figura 2. Selección por torneos. Fuente: https://jeongchul.tistory.com/573

La mutación es muy importante en los algoritmos genéticos. Sin ella, estos algoritmos convergerían muy rápidamente a un mínimo local. Además, la mutación permite evitar la pérdida de información genética ya que puede recuperar alelos perdidos mediante cruces. También permite aleatorizar las búsquedas y así poder encontrar nuevos mínimos locales. Normalmente en las primeras fases se potencia más la exploración (mutación) y después la explotación (cruce), aunque para salir de un mínimo se puede dar algún arreón de mutación en algún momento concreto. En cualquier caso, la probabilidad de mutación puede ser variable bajo algún criterio que fije el diseñador del algoritmo.

Y, por último, la función de ***fitness*** que es la que nos determina cómo de bueno es un individuo y la que guía la búsqueda. Normalmente se busca minimizar la función, aunque también se puede maximizar en función de cómo lo consideres.

Ejemplo

Vamos a ver un ejemplo de cómo codificar el problema del viajero usando algoritmos genéticos y realizaremos una iteración de este.

Imaginemos que queremos codificar cuatro ciudades, podemos utilizar 2 bits.

Codificación
00
01
10
11

La codificación de un individuo puede ser 2 bits y un individuo codificarse así:

1 => 3 => 2 => 4 = 00100111

Generamos una población aleatoria:

10101010

11011001

00010101

01010101

Después valoramos con la función de fitness (la distancia recorrida):

10101010 => 3

11011001 => 2

00010101 => 4

01010101 => 2

Si usamos ruleta decimos que la probabilidad de ser elegido será más alta para el individuo 2 y 4. Supongamos que se eligen ambos para el cruce. Se cruzan con cruce simple.

11011001 x 0101**0101** => 11010101

Se muta con una probabilidad 11010101 => 1101011**1**

Se repite hasta tener cuatro nuevos individuos.

Se repite el proceso hasta que no se mejore más.

Esta codificación tiene un problema y es que permite visitar la misma ciudad dos veces. Estos individuos serían erróneos. ¿Qué sucede con ellos? Se les pueden penalizar fuertemente con el fitness o impedir que se generen. Pero la solución más eficiente es codificar de otra forma el problema.

Podemos codificar el problema de forma que ordenamos las ciudades de menor a mayor según un criterio arbitrario. Cuando codificamos un número, en binario, codificamos la posición de la ciudad en la lista. De forma que 0101 no significa que vayamos de la ciudad 2 a la 2 sino de la 2 a la 3 (la numero 2 de la lista una vez que hemos quitado la segunda). De esta forma podemos codificar el sistema de forma más eficiente (con menos bits) y evitar individuos erróneos. Piensa que cuanto más grandes sean los individuos, el espacio de búsqueda será mayor.

Otra de las cuestiones que debemos tener en cuenta es que, en el cruce, normalmente va implícito una relación entre las diferentes posiciones de los genes. Si la codificación no mantiene esa relación de posición, el cruce guía peor la búsqueda y esta se vuelve más aleatoria.

El algoritmo se muestra en la siguiente Figura 3:

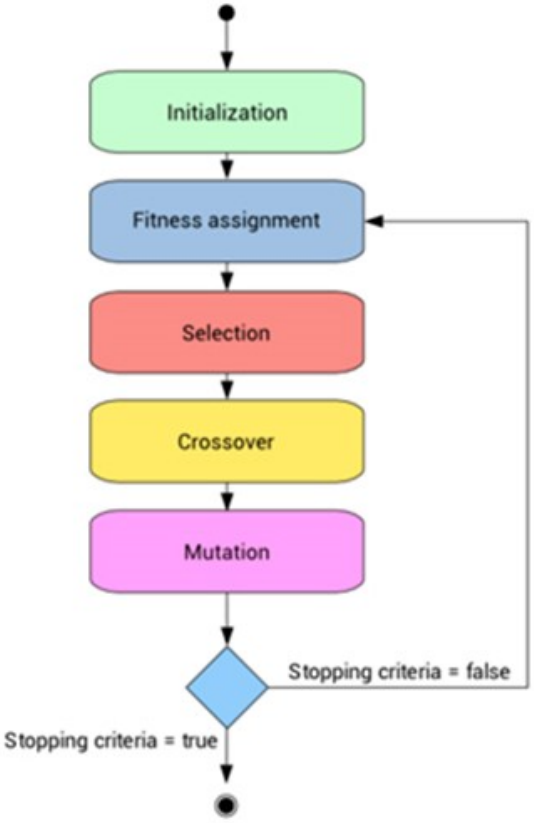


Figura 3. Diagrama de un algoritmo genético.

Estrategias evolutivas

Las estrategias evolutivas son similares a los algoritmos genéticos. La principal diferencia es que **opera con una codificación de valores continuos.** Esto implica que la mutación debe cambiar ya que no existe el concepto de mutación discreta, es decir, no podemos elegir una de las diferentes alternativas del valor del gen, ya que ahora hay infinitos valores. La mutación va a seguir una distribución normal de probabilidad. Esto hace que la heurística de búsqueda se establezca precisamente en el operador de mutación y no en el cruce como en los algoritmos genéticos.

Se codifican los individuos con un vector de codificación

x_i y un vector de varianzas

σ_i que indica lo alejado que se encuentra de la solución del problema. Pero no

sabemos cuál es la solución óptima, por lo tanto, este vector de varianzas en realidad son predicciones de la distancia al óptimo. Este vector también se evoluciona, por lo tanto, es de esperar que esta predicción mejore con el tiempo.

Los diferentes algoritmos de estrategias evolutivas siguen la notación $(\mu/\rho,\alpha)$ donde:

μ Tamaño de la población.

ρ número de padres que se seleccionan para recombinarse.

α número de individuos en la descendencia.

Lo más simple, el modelo (1+1) - EE

Este es el **algoritmo más básico y es un método de escalada puro.** Un individuo se denota de la siguiente forma $P = x,\sigma$ donde x y σ son el vector de codificación y el de varianza.

En este algoritmo se genera una población intermedia mutando al individuo y después se evalúa. Si es mejor se reemplaza y si no se mantiene el actual. Como solo hay un individuo no hay cruce.

La mutación del vector de soluciones se realiza usando una distribución normal de media 0 y varianza σ .

$$x_i' = x_i + \sigma_i' \cdot N(0,1)$$

Esto tiene sentido debido a que la varianza al principio será muy grande (la distancia a la solución) y los saltos que se produzcan serán más grandes. A medida que nos vayamos acercándonos a la solución, los saltos son más pequeños porque estamos más cerca (exploración vs. explotación).

Para actualizar la varianza se usa la regla de 1/5. Dicha regla lleva una estadística del número de veces que la población ha cambiado y se ha encontrado una mejor solución. Para ello establecemos un valor del número de generaciones que vamos a tener en cuenta y cuántas veces se ha cambiado en esas generaciones. Esto se denota como

γ_s^t donde *t* es el número de veces que se ha realizado reemplazo y *s* el número de generaciones anteriores que tenemos en cuenta. Si la ratio ente ellas varia de 1/5 se incrementa o se decrementa el valor de la varianza siguiendo la siguiente regla:

$$\sigma' = \sigma \cdot const_d \leftrightarrow \gamma_s^t < 1/5$$

$$\sigma' = \sigma \cdot const_i \leftrightarrow \gamma_s^t > 1/5$$

Donde

$const_d \, y \, const_i$ son dos valores. El primero menor que 1 y el segundo mayor que 1.

De esta forma vamos evolucionando el algoritmo hasta que no cambie sustancialmente en una iteración o hasta que encuentre la solución (el error sea admisible).

Este es el ejemplo más simple, pero se pueden extender a modelos con múltiples individuos.

Modelos con múltiples individuos

Estos modelos comparten gran parte de las ideas del ejemplo simple, pero incorporan el operador de cruce. El cruce de dos individuos se realiza haciendo la media aritmética de cada uno de ellos. También hay que cruzar el vector de varianzas, pero este se debe hacer calculando la raíz cuadrada de la suma de las varianzas de los dos individuos.

$$x' = \frac{1}{2} \cdot (x1 + x_2)$$

$$\sigma' = \sqrt{\sigma_1 + \sigma_2}$$

El ámbito de aplicación de estos algoritmos es el mismo que los algoritmos genéticos, y se utilizan en vez de estos cuando el sistema a optimizar maneja valores continuos. Así pues, cualquier aplicación realizada con algoritmos genéticos con números reales puede aplicarse a las estrategias evolutivas. Se han demostrado especialmente útiles por ejemplo en el entrenamiento de redes de neuronas, modificando sus pesos mediante estrategias evolutivas.

Programación genética

La programación genética es un algoritmo evolutivo derivado de los algoritmos genéticos propuestos en 1989 por John Koza.

La idea de la programación genética es **evolucionar programas completos.** Para poder hacer esto, debemos crear un lenguaje de programación más sencillo que nos permita encontrar una solución a un problema concreto que queremos obtener. De esta forma a la programación genética debemos darle los operadores y funciones necesarios que estimemos que va a necesitar el algoritmo de la solución. Por ejemplo, si vamos a resolver una ecuación, pues introduciremos operadores, variables, constantes, etc.

La idea del algoritmo consiste en generar una población de programas que intentan resolver el problema que se plantea. En la figura se muestra un ejemplo de un individuo.

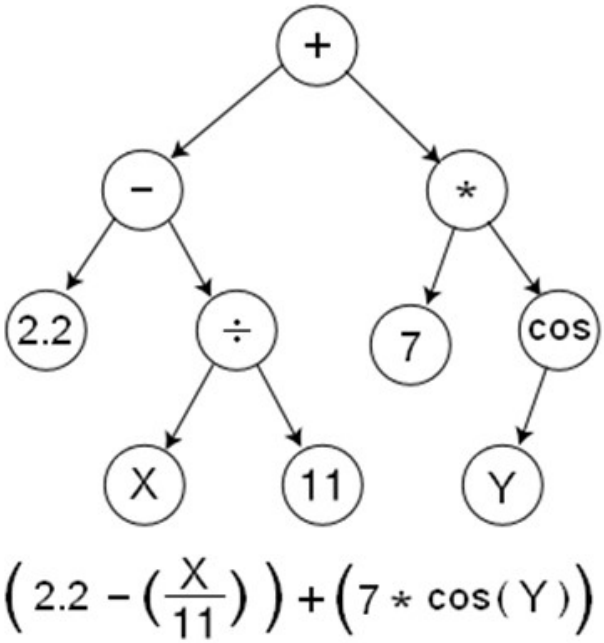


Figura 4. Ejemplo de un individuo que calcula una función usando programación genética. Fuente: https://es.wikipedia.org/wiki/Programaci%C3%B3n_gen%C3%A9tica

Si queremos hacer un programa que calcule una ecuación, el algoritmo funcionaría de la siguiente forma:

- Se genera un conjunto de ecuaciones aleatorias en forma de árbol (notación polaca o polaca inversa, por ejemplo).
- Cada ecuación del conjunto de ejemplos contendrá un valor para las variables que se necesitan.
- Se sustituyen dichos valores en la ecuación representada por el individuo a evaluar, y así se obtiene un valor de la ecuación.
- Se compara el valor con el esperado y si es el mismo, el individuo resuelve la ecuación y si no es así, la diferencia se utilizará para el cálculo del fitness del individuo.

Se aplican las mismas ideas que en los algoritmos genéticos. Por ejemplo, el **cruce consistirá en coger un subárbol de cada individuo e intercambiarlos.**

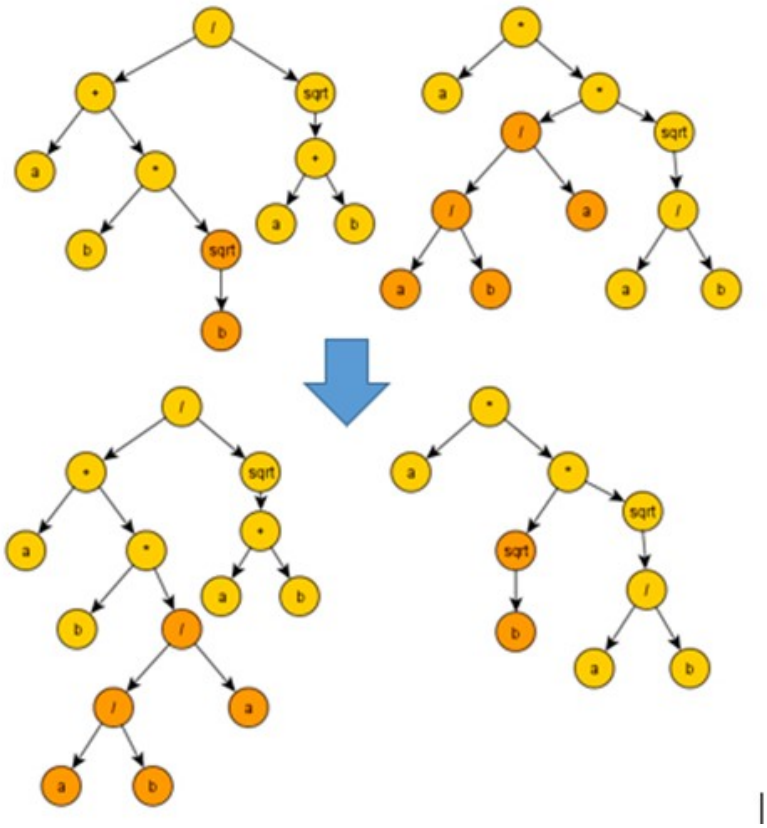


Figura 5. Cruce en programación genética.

La **mutación** consiste en cambiar un operador por otro o un valor o variable por otro. También se puede mutar un subárbol generándolo aleatoriamente.

Redes de neuronas

Las redes de neuronas son otros de los sistemas más exitosos con inspiración biológica que se utilizan actualmente. Ya hablamos de ellas anteriormente, por lo que no nos extenderemos demasiado aquí. Simplemente recordaremos su inspiración biológica, basada en la organización del cerebro animal, donde **la potencia de**

cálculo principal se encuentra en las reacciones químicas que se producen en las interconexiones de las células o neuronas. Así pues, las redes de neuronas artificiales llevan el cálculo a las interconexiones de las neuronas y no tanto a estas, que simplemente procesan mínimamente la señal recibida con funciones simple como la sigmoidal.

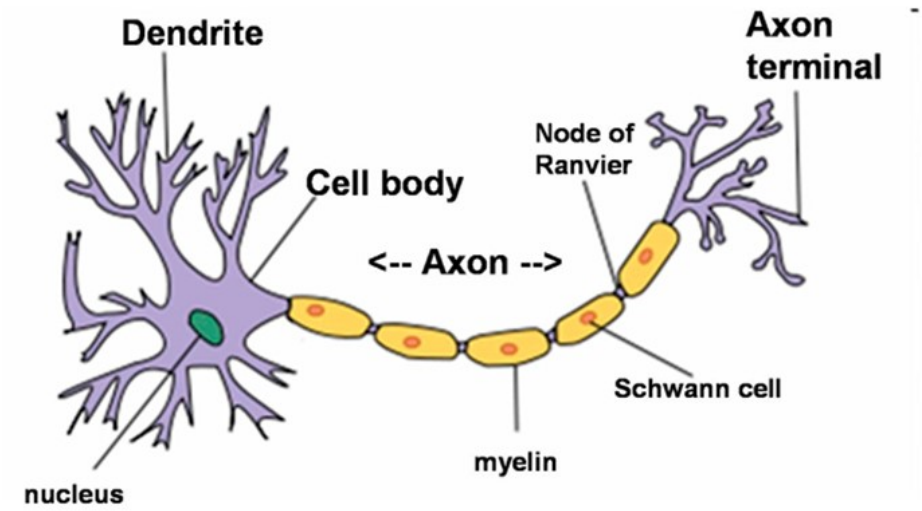


Figura 6. Esquema de una neurona.

Su uso, como sabemos con el éxito del *deep learning,* está muy extendido actualmente y puede utilizarse en multitud de dominios, debido a que se le considera un aproximador universal de funciones.

Sistema inmunitario artificial

Los sistemas inmunitarios artificiales (AIS) **son una clase de máquinas de aprendizaje basado en reglas computacionalmente inteligentes, inspiradas en los principios y procesos del sistema inmunitario de los seres vivos.** Esos algoritmos son modelados a partir de las características de aprendizaje y memorización del sistema inmunitario y están enfocados en la resolución de problemas, de forma similar a los algoritmos genéticos.

Las técnicas comunes están inspiradas en teorías inmunológicas que explican el

funcionamiento y el comportamiento del sistema inmunitario adquirido. Existen diferentes técnicas, pero nos centramos en la más común, el ***clonal selection algorithm*** **o algoritmo de selección clonal.**

Los algoritmos de selección clonal son una c**lase de algoritmos inspirados en la teoría de selección clonal de la inmunidad adquirida,** que explica cómo los linfocitos B y T mejoran su respuesta ante antígenos con el tiempo mediante la maduración de la afinidad. Son interesantes porque pretenden obtener las características del sistema inmunitario que son:

- Tolerante a fallos.
- Robusto.
- Dinámico.
- Descentralizado.
- Adaptativo.
- Autoprotegido.
- Diverso.

¿Cómo funciona el sistema inmune?

Cuando un animal es expuesto a un antígeno, algunas de sus células derivadas de la médula ósea (linfocitos B) responden produciendo anticuerpos. Cada célula secreta solo un tipo de anticuerpo, que es específico para el antígeno detectado. Al unirse a estos anticuerpos (receptores), y con una segunda señal de células accesorias, como las células T, el antígeno estimula al linfocito B para que prolifere (se divida) y

madure en células secretoras de anticuerpos terminales. A diferencia de los linfocitos B, estas células no se pueden clonar. A estas células terminales se la denomina **células de plasma.** Los linfocitos T son los reguladores de esta respuesta de los linfocitos B. En realidad, son los que detectan a los antígenos cuando estos han infectado alguna célula o han sido procesados por macrófagos mediante un patrón de afinidad. Cada linfocito T responde a un antígeno concreto.

La **selección clonal** hace una analogía de las soluciones candidatas a anticuerpos buscando afinidad. La **afinidad se mide como la semejanza con un patrón antígeno** mediante la evaluación de una función, similar a la función de *fitness*. De forma que el antígeno en realidad es la función de afinidad. Los anticuerpos con mejor afinidad se someten a una clonación proporcional a dicho valor de afinidad y a la mutación de dichos clones. Esta mutación es mayor cuanto menor es la afinidad. Una vez realizada la clonación, estos compiten con la población de anticuerpos existente por ser miembro de la próxima generación.

El **algoritmo** sería el siguiente:

- Se genera un conjunto de soluciones *P* candidatas aleatorias.
- Se selecciona los *n* mejores individuos en base a una función de afinidad con el patrón antígeno presentado (el problema a resolver).
- Se clonan los *n* individuos un número de veces por cada individuo seleccionado. Este número es proporcional a la afinidad de cada individuo. Se puede utilizar la siguiente ecuación donde β es un parámetro ajustable.

$$N_i = \beta \cdot \frac{N}{affinity(i)}$$

- Se mutan en proporción a la afinidad. Si hay menor afinidad, la mutación es más agresiva. Se puede utilizar la siguiente ecuación para calcular la tasa de mutación donde *ρ* es un parámetro ajustable.

$$mutation\,rate\,(\alpha) = \exp\,(-\rho \cdot \frac{affinityMin}{affinityMax})$$

- El número de clones se puede realizar siguiendo una distribución normal de media 0 y desviación típica
  α .
- Se evalúan de nuevo cada anticuerpo de los generados por clonación.
- De entre las nuevas soluciones generadas se seleccionan los *n* mejores y se remplazan en la población original.
- Los *d* anticuerpos con menor afinidad son sustituidos por otros anticuerpos generados aleatoriamente.
- El proceso continúa hasta alcanzar el criterio de parada.

Algunas **aplicaciones** sobre de estos algoritmos son:

- Detección de intrusos.
- Extracción de características.

- Problemas de optimización.

Optimización multiobjetivo

En un problema de optimización **se tratará de encontrar una solución que represente el valor óptimo para una función objetivo.** Pero en ciencias e ingeniería se dan, en bastantes ocasiones, problemas que requieren la optimización simultánea de más de un objetivo. A esto se le denomina **optimización multiobjetivo.**

Estos problemas se resuelven principalmente con la **optimización de frente de Pareto.** El frente de Pareto es un concepto proveniente de las ciencias económicas que consiste en **realizar una asignación de recursos a los diferentes objetivos de forma que no se pueda mejorar uno sin perjudicar otro.** A esto se le denomina frente de Pareto optimo o eficiente. Existen multitud de algoritmos que buscan resolver este problema. Uno de los más comunes es NSGA II.

NSGA II (*Non dominated sorting genetic algorithm)*

Se deben definir unas funciones objetivas que se pretenden maximizar. Normalmente ambas funciones son contrarias, es decir, si mejoramos una empeoramos otra. Por ejemplo, una función de beneficio y una función de coste.

Se denomina **dominancia de una solución sobre otra** si cumple las siguientes condiciones:

- La primera solución es mejor que la segunda en todos los objetivos.
- La primera solución es estrictamente mejor que la segunda en al menos uno de los objetivos.

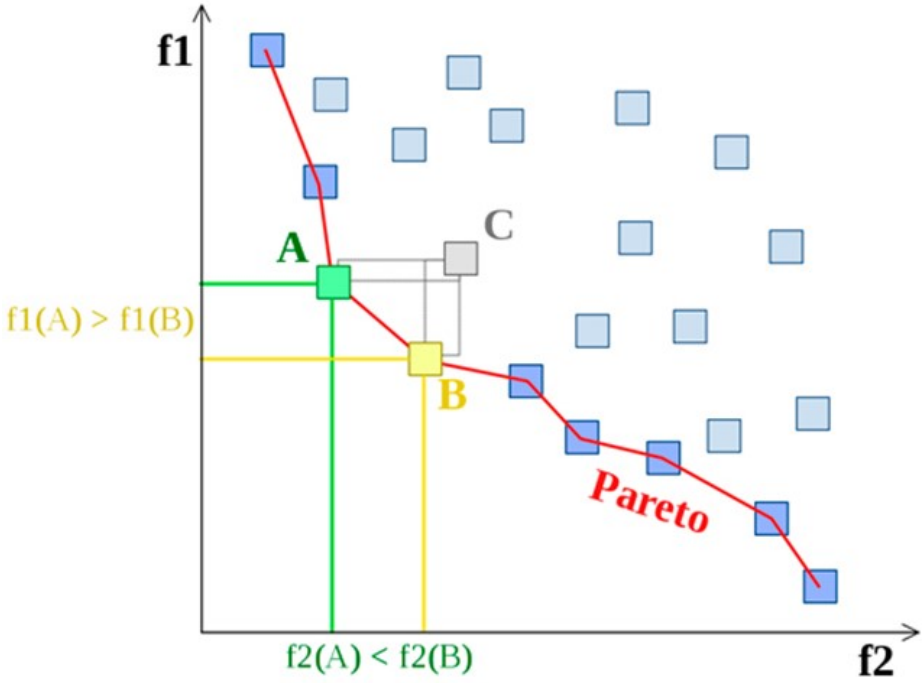


Figura 7. Frente de Pareto. La opción C no está en el frente de Pareto ya que A y B lo dominan. Estos a su vez no son dominados por ningún otro, consecuentemente están en la frontera. Fuente: https://es.wikipedia.org/wiki/Eficiencia_de_Pareto#/media/Archivo:Front_pareto.svg

Para la optimización de un problema con múltiples objetivos no existe una única solución, sino un conjunto de soluciones no dominadas, en la cual no existen soluciones mejores en todos los objetivos.

NSGA II funciona muy similar a un algoritmo genético normal. Tiene las mismas etapas de selección cruce y mutación. La diferencia es que se debe calcular el frente de Pareto de la solución frente a las diferentes funciones de coste y beneficio. Y los criterios de selección se realizan en función de aquellos individuos que generan un frente de Pareto más eficiente sin generar soluciones no dominadas. La nueva población es generada a partir de las configuraciones de los frentes no dominados. Esta nueva población empieza a ser construida con el mejor frente no dominado (F1), continúa con las soluciones del segundo frente (F2), tercero (F3) y así

sucesivamente. Como la población es de tamaño 2N, y solamente existen N configuraciones que conforman la población descendiente, no todas las configuraciones podrán ser acomodados en la nueva población. Aquellos frentes que no pueden ser acomodados desaparecen.

Este proceso se repite hasta que se obtiene un individuo que satisface los criterios planteados como criterio de parada.

11.4. Sistemas emergentes

Los sistemas emergentes son **sistemas complejos compuestos normalmente por elementos con un comportamiento muy simple, pero que realizan en conjunto un comportamiento poco esperado.** Algunas técnicas empleadas en computación se han basado en este tipo de sistemas para conseguir solucionar problemas complejos. Se caracterizan por resolver problemas, al menos en apariencia, espontáneamente; es decir, sin recurrir a una inteligencia de tipo centralizado o jerarquizado (descendente), sino de forma ascendente, desde la base, a partir de masas de elementos relativamente no inteligentes.

Existen muchas aproximaciones de este tipo para modelizar comportamientos y simularlos computacionalmente como por ejemplo para **simular el comportamiento de manadas *(flock)* o de multitudes se pueden utilizar sistemas de enjambres *(swarm)*** en los que cada individuo se modela con simples reglas que afectan solo a su entorno local. A parte de estas simulaciones que están inspiradas en cómo se comportan los enjambres y manadas de aves, mamíferos o insectos y cómo evitan, por ejemplo, chocarse, existe un grupo de algoritmos que utilizan estos conceptos más allá de la modelización de sistemas sino para resolver problemas de optimización. A continuación, veremos algunos de ellos.

Colonias de hormigas

Los algoritmos basados en colonias de hormigas son algoritmos bioinspirados, que se aprovechan de una característica interesante de las colonias de hormigas. Y es que son capaces de encontrar el camino más corto entre dos puntos de forma automática.

Figura 8. Las hormigas son capaces de encontrar el camino más corto. Fuente: http://ornewspaper.blogspot.com/2011/11/la-investigacion-de-operaciones-en-la.html

El sistema funciona a través del concepto de feromona. Las hormigas artificiales son unas unidades o pequeños agentes que se mueven de forma aleatoria por el espacio de soluciones. Al moverse van dejando una feromona a su paso.

La sustancia emitida puede ser detectada por otras hormigas:

- La hormiga elige el camino proporcional a la feromona que esté en dicho camino.
- La feromona del camino se disipa con el tiempo, y si no se vuelve a generar desaparece.
- Los caminos poco frecuentados casi no tienen posibilidad de ser utilizados, a menos que se encuentre comida y se repita el proceso.

Este comportamiento elemental explica como las hormigas son capaces de seguir caminos cortos hacia lugares relativamente alejados. También son capaces de adaptarse a cambios producidos en el entorno, por ejemplo, a encontrar un nuevo camino si hay un obstáculo.

Lo interesante de las colonias de hormigas es que encontrar el camino más corto

entre dos puntos es una **propiedad emergente** de la propia interacción con el entorno.

Llevado esto al terreno computacional, una hormiga no es más que un pequeño agente que se mueve de forma semialeatoria. Tiene mayor probabilidad de moverse hacia una posición donde antes ha pasado una hormiga que haya dejado su feromona depositada, pero no lo garantiza. Esta aleatoriedad le da al algoritmo cierta capacidad de exploración. También hace que pueda adaptarse las soluciones si el entorno ha cambiado. Las hormigas, por tanto, solo utilizan información local para tomar las decisiones y la memoria colectiva que las comunica es la feromona.

Para que el algoritmo funcione debemos **discretizar el espacio de búsqueda.** La emisión de feromona de nuestra hormiga virtual puede ser global en todo el camino o no. Las hormigas tienen también una cierta memoria. Dicha memoria impide que la hormiga vuelva sobre sus pasos.

La feromona que van dejando las hormigas se va disipando con las sucesivas iteraciones. Esto implica que se van olvidando los caminos por donde no pasan las hormigas (se van olvidando soluciones antiguas subóptimas).

Cada hormiga tiene una simple tabla de decisiones que tiene una entrada por cada nodo al que pueda transitar dentro del espacio de estados discreto. Esta entrada es inversamente proporcional al coste de alcanzar el nodo y directamente proporcional a la feromona del tramo. Se puede usar la siguiente ecuación donde $\alpha y \beta$ son parámetros de ajuste.

$$probabilidad\,(tramo\,iaj)\,a_{ij} = \frac{feromona_{ij}^{\alpha} \cdot \frac{1}{coste_{ij}}^{\beta}}{\sum I \in Nodos_i = feromona_{iI}^{\alpha} \cdot \frac{1}{coste_{iI}}^{\beta}}$$

Es decir, la probabilidad de elegir un nodo es el producto entre la feromona del nodo y la inversa del coste de dicho nodo, ajustado por dos parámetros $\alpha y \beta$ entre el acumulado de todas las feromonas y los costes de todos los posibles

nodos adyacentes al nodo actual donde está la hormiga.

El rastro de la feromona que deja la hormiga en cada tramo se puede calcular con la siguiente ecuación:

$$\Delta\tau_{ij} = \begin{cases} \frac{1}{Longitud(C)} si\, i,j \epsilon C \\ 0\, en\, caso\, contrario \end{cases}$$

La disipación de la feromona con el tiempo se calcula de la siguiente forma:

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}$$

Donde ρ es un parámetro entre 0 y 1 que se le conoce como coeficiente de evaporación.

Podamos añadir **elitismo** si establecemos que la hormiga que ha hecho el camino más corto agrega más feromona.

Aplicaciones de las colonias de hormigas

Los algoritmos de optimización de colonias de hormigas son aplicados en muchos algoritmos de **optimización combinatorio.** Se han adaptado a problemas dinámicos en variables reales, problemas estocásticos, programación paralela y multiobjetivo. Estos algoritmos tienen una ventaja sobre algoritmos genéticos cuando el grafo puede cambiar su estructura de manera dinámica. Los algoritmos de colonias de hormigas pueden seguir ejecutándose de forma permanente y adaptarse a la nueva estructura del grafo. Esto es de mucho interés en los campos de enrutamiento de redes y en los sistemas de transportes urbanos. También se han utilizado para generar arte.

Optimización de enjambres de partículas

Es una **metaheurística que evoca el comportamiento de las partículas en la naturaleza.** En un principio fueron concebidos para elaborar modelos de conducta

social, como por ejemplo los movimientos descritos por los seres vivos en una bandada de aves o en un banco de peces. Sin embargo, se vio que estos algoritmos eran adecuados para resolver problemas de optimización.

Estos algoritmos permiten optimizar un problema a partir de una población de soluciones candidatas, que denominamos partículas. Estas partículas se mueven dentro del espacio de búsqueda según una serie de reglas matemáticas sencillas que tienen en cuenta la posición y la velocidad de la partícula. El movimiento de cada partícula depende de su mejor posición obtenida, así como de la mejor posición global hallada en todo el espacio de búsqueda. A medida que se descubren nuevas y mejores posiciones, estas pasan a orientar los movimientos de las partículas. El proceso se repite con el objetivo, no garantizado, de hallar en algún momento una solución lo suficientemente satisfactoria.

Así pues, cada partícula recuerda su mejor posición y se desplaza con cierta aleatoriedad por el espacio, pero siempre en torno a su mejor posición. Con el tiempo las partículas convergen en los mínimos locales.

Como tienen cierta aleatoriedad, pueden salir de los mínimos locales. Se puede jugar con el grado de aleatoriedad en el cambio de la dirección de la partícula.

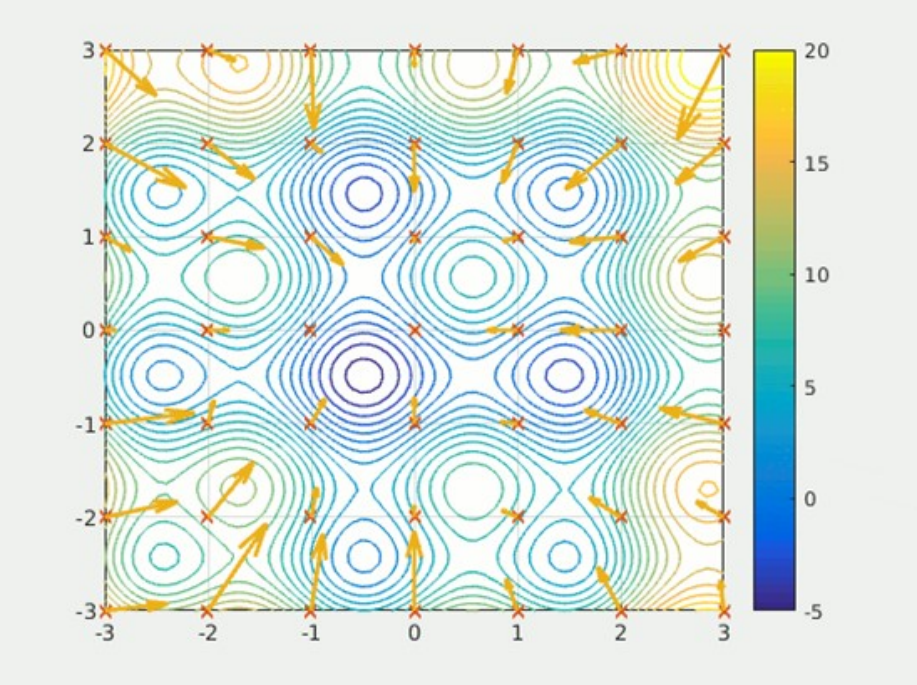


Figura 9. Ejemplo de optimización por enjambre de partículas. Fuente:

https://en.wikipedia.org/wiki/Particle_swarm_optimization#/media/File:ParticleSwarmArrowsAnimation.gif

11.5. Proyectos de investigación sobre computación bioinspirada

Los proyectos de investigación en computación bioinspirada son uno de los más activos dentro de la investigación actual. Son áreas candentes en esta investigación e l *deep learning,* los algoritmos genéticos, colonias de hormigas, sistemas de optimización de partículas y enjambres para drones, robótica, etc., Los problemas donde se pueden aplicar estos sistemas de algoritmos son casi infinitos, pero vamos a concretar los más importantes describiendo algunos de ellos:

- Todo tipo de clasificación automática.
- Reconocimiento de imágenes.
- Generación artística artificial.
- Simulación de vida artificial.
- Programación de agentes automáticos.
- Optimización.
- Sistemas dinámicos.
- Sistemas de optimización multiobjetivo.

Los proyectos de computación evolutiva y optimización suelen requerir de grandes cantidades de computación, por lo que es muy recomendable acudir a soluciones de terceros. Están en una situación similar a los algoritmos de *machine learning.* Aquí la dependencia de los datos es menor que en *machine learning,* ya que la mayoría de estos algoritmos se emplean en modelos de optimización más que en la de ciencia de datos. Aunque se han visto soluciones híbridas de *machine learning* y metaheurísticas bioinspiradas en multitud de proyectos.

Ideas clave

Las **principales publicaciones** de este medio son:

- *Computers & Structures.*
- *Journal of Global Optimization.*
- *IEEE Transactions on Evolutionary Computation.*
- *Expert System with Application.*
- *Evolutionary Computation.*

11.6. Referencias bibliográficas

Davis, L. (1991). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold Company.

Gestal, M., Rabuñal, J. R., Dorado, J. Pazos, A. y Rivero, D. (2010). *Introducción a los algoritmos genéticos y la programación genética.* A Coruña: Universidade da Coruña.

Algoritmos genéticos y computación evolutiva

Marczyk, A. (2004). Algoritmos genéticos y computación evolutiva. *the-geek.org.* Recuperado de https://the-geek.org/docs/algen/algen.html

Libro web donde puede expandir los conocimientos de algunos de los algoritmos que exponemos en este tema.

¿Qué son los algoritmos genéticos?

Brunvelop. (2 de abril de 2020). *Esta I.A. Aprende a caminar con algoritmos genéticos* [Vídeo]. YouTube. https://www.youtube.com/watch?time_continue=27&v=06iwEZokIGk&feature=emb_title

Vídeo explicativo de cómo funciona un algoritmo genético.

Algoritmo Inmune de Selección Clonal para el problema de Job Shop Scheduling

Gómez, W. A. (agosto de 2013). Algoritmo Inmunse de Selección Clonar para el problema de Job Shop Scheduling. Conference *IV Congreso Internacional de Computación e Informática del Norte de Chile.* Recuperado de https://www.researchgate.net/publication/258110429_Algoritmo_Inmune_de_Seleccion_Clonal_para_el_problema_de_Job_Shop_Scheduling#:~:text=El%20algoritmo%20de%20selecci%C3%B3n%20clonal%20%28CLONALG%29%20%5B16%5D%20se,la%20cu%20al%20implica%20la%20selecci%C3%B3n%20de%20anticuerpos

Artículo donde se ve una aplicación de la selección clonal de sistemas inmunes en un problema de planificación de tareas.

Drawing with Ants: Generative Art with Ant Colony Optimization Algorithms

LOBSTERS. (18 de enero de 2020). Drawing with Ants: Generative Art with Ant Colony Optimization Algorithms. *colabug.com.* Recuperado de https://www.colabug.com/2020/0118/6871092/

Artículo donde se explica cómo generar arte usando colonias de hormigas.

1. ¿Qué es la computación biológica?

   A. Es el área de conocimiento que estudia cómo podemos utilizar elementos de naturaleza biológica para obtener mejores resultados en problemas de computación.

   B. Es el uso de técnicas de computación en el estudio de la biología.

   C. Es el estudio de la biología haciendo una analogía con la computación.

   D. Ninguna de las anteriores.

2. En la teoría de la selección natural, la presión selectiva sirve para:

   A. Que se detenga la evolución.

   B. Que la evolución se produzca, ya que es el mecanismo por el cual las células transmiten la herencia genética.

   C. Guiar a la evolución para adaptar los individuos al entorno.

   D. Favorece la mutación de los individuos.

3. ¿Qué es la computación evolutiva?

   A. Es un conjunto de técnicas que permite a las máquinas aprender de forma supervisada y usar la solución de cada ejemplo como parte del aprendizaje.

   B. Es la rama de la IA que busca resolver problemas de optimización, inspirándose en mecanismos evolutivos.

   C. Es la rama de la IA que busca resolver problemas de clasificación, inspirándose en mecanismos evolutivos.

   D. Ninguna de las anteriores.

4. En los algoritmos genéticos, la mutación sirve para:

A. Aportar variabilidad genética.

B. Buscar nuevas soluciones y salir de mínimos locales.

C. Mantener o recuperar los alelos perdidos.

D. Todas las anteriores son correctas.

5. Si hay más presión selectiva en un algoritmo genético, ¿qué sucede?

A. Que disminuye la diversidad genética.

B. Que aumenta la diversidad genética.

C. Que la solución que se obtenga será más óptima.

D. Ninguna de las anteriores.

6. Indica el motivo por el cual un algoritmo genético se puede estancar prematuramente:

A. Que apenas haya presión selectiva.

B. Que apenas haya diversidad genética.

C. Que esté estancado en un mínimo local y el mínimo global esté muy lejos, siendo la tasa de mutación no lo suficientemente grande.

D. Todas las anteriores.

7. ¿Cómo configurarías un algoritmo genético para maximizar los resultados?

A. Al principio pondría tasas de mutación baja para explotar y con el tiempo pondría tasas de mutación alta para explorar.

B. No pondría mutación.

C. Al principio pondría tasas de mutación alta para explorar y con el tiempo pondría tasas de mutación baja para explotar.

D. Ninguna de las anteriores.

8. Las estrategias evolutivas:

A. Son similares a los algoritmos genéticos pero los datos que procesan son discretos.

B. Son similares a los algoritmos genéticos pero los datos que procesan son continuos.

C. Sirve para clasificación y para regresión.

D. Todas las anteriores son correctas.

9. En cuanto a la optimización multiobjetivo:

A. Se calcula el frente de Pareto que determina la región de decisión óptima que hace que no se mejore ni empeore ninguna de las funciones objetivo para cada uno de los posibles valores del dominio.

B. Pretende optimizar dos algoritmos simultáneamente.

C. Pretende aprender un único valor que optimice los múltiples objetivos buscados.

D. Todas las anteriores son correctas.

10. ¿Qué es un sistema emergente?

A. Son sesgos no esperados en los comportamientos aprendidos por los algoritmos de *machine learning.*

B. Sistemas complejos compuestos por elementos muy simples de cuyo comportamiento no se puede extrapolar el comportamiento final del conjunto.

C. Un sistema que es capaz de extraer soluciones que otros sistemas no permite extraer.

D. Todas las anteriores son falsas.