

Guía de Estudio para Algoritmos Bioinspirados

1. Introducción

Los algoritmos bioinspirados son técnicas de optimización y búsqueda basadas en principios y mecanismos observados en sistemas biológicos. Estos algoritmos aprovechan la eficiencia y adaptabilidad de la naturaleza para resolver problemas complejos en diversas áreas como la ingeniería, la informática y las ciencias naturales.

Algunos de los algoritmos más importantes de este enfoque son:

Algoritmos Genéticos (GA)

Principio: basado en la selección natural y la genética.

Componentes Clave: población, cromosomas, selección, cruza y mutación.

Aplicaciones: optimización, problemas de búsqueda.

Programación Genética (PA)

Principio: basado en la selección natural y la genética.

Componentes Clave: población, árboles sintácticos, selección, cruza y mutación.

Aplicaciones: modelado.

Algoritmo de Enjambre de Partículas (PSO)

Principio: basado en el comportamiento social de los animales, como aves y peces.

Componentes Clave: partículas, velocidad, posición, líder global o local.

Aplicaciones: optimización continua y discreta.

Algoritmos de Colonia de Hormigas (ACO)

Principio: imitan el comportamiento de las hormigas en busca de comida.

Componentes Clave: feromonas, camino más corto, evaporación de feromonas.

Aplicaciones: problemas de ruta, como el problema del viajante de comercio.

Algoritmos de Colonia de Abejas (ABC)

Principio: Imitan el comportamiento de las abejas en busca de comida.

Componentes Clave: Feromonas, camino más corto, evaporación de feromonas.

Aplicaciones: optimización continua y discreta.

La Figura 1 muestra un esquema más completo de diferentes enfoques que se encuentran dentro del campo de los algoritmos bioinspirados.

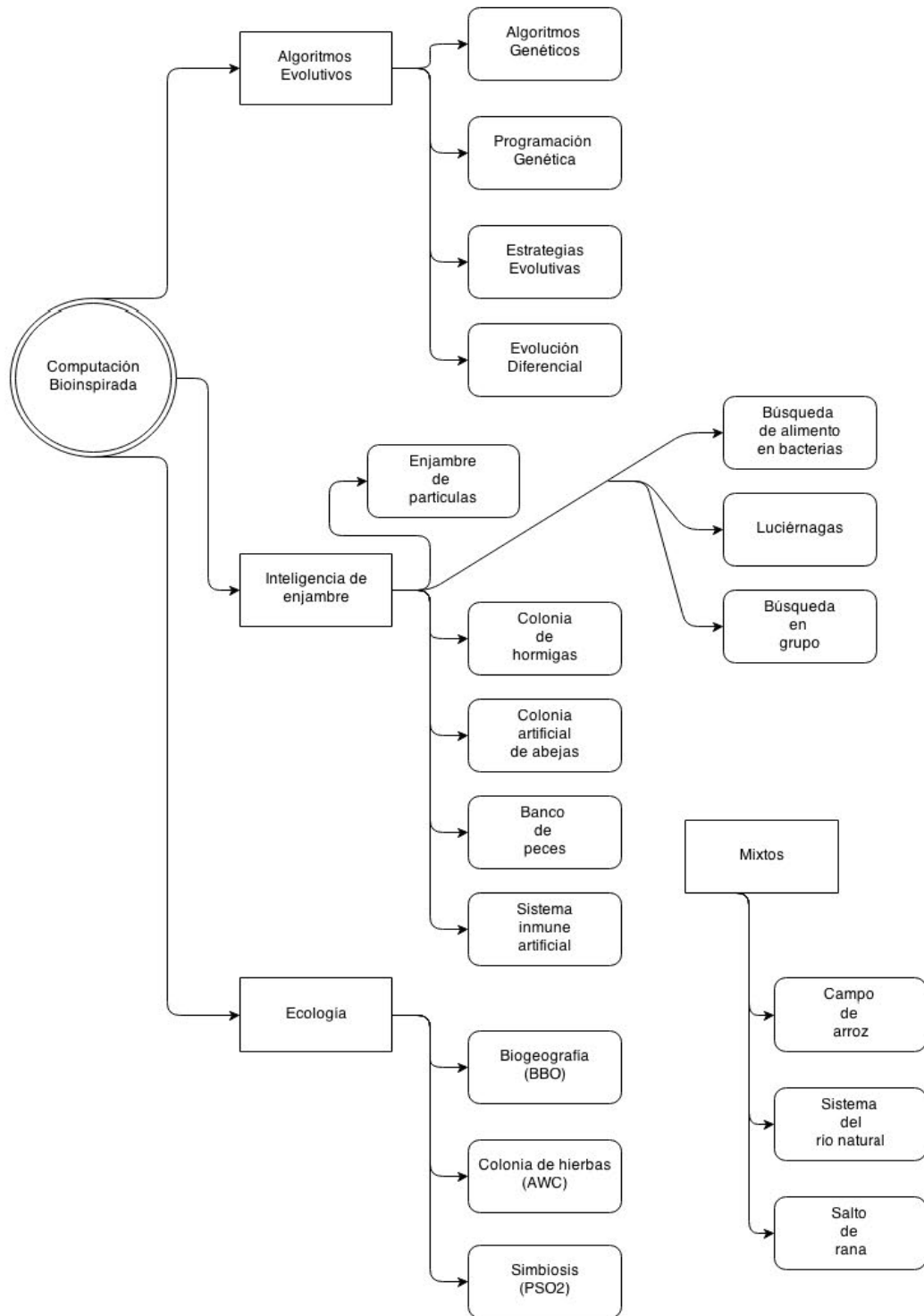


Figura 1. Esquema General de Algoritmos Bioinspirados

2. Optimización y Espacios de Búsqueda

En la ingeniería, los problemas a los que nos enfrentamos en la realidad pueden ser formulados matemáticamente en forma de problemas de optimización, en los que se busca encontrar la mejor solución posible al mismo, entre todas sus soluciones disponibles. En general, optimizar significa que se trata de buscar la mejor solución a un problema, que puede estar sujeta a restricciones o no, algunos ejemplos son:

- Maximizar ganancias dado un número limitado de recursos.
- Minimizar emisiones contaminantes sin afectar la movilidad.
- Maximizar o minimizar una función.



Figura 2. Ejemplos de funciones con máximo o mínimo

Definición de optimización

Consideramos el problema de optimización *minimizar o maximizar* $f(x)$ donde x es un vector n -dimensional de variables independientes

$$x = (x_1, x_2, \dots, x_n) \in R^n$$

Sujeto a restricciones

$$\begin{aligned} h(x) &= 0 \\ g(x) &\leq 0 \end{aligned}$$

Función objetivo: la función f que se debe minimizar o maximizar.

Variables de diseño: el vector $x = (x_1, x_2, \dots, x_n) \in R^n$ son los parámetros que el algoritmo de optimización puede elegir para minimizar la función objetivo. Estas variables deberían ser independientes entre si.

Restricciones: son funciones sobre las variables de diseño que se quieren restringir. Cuando la función debe ser igual a un valor es una restricción de igualdad. Cuando la función debe ser mayor o igual a un valor es una restricción de desigualdad

Como primeras herramientas para intentar dar respuesta a este tipo de problemas se utilizaron los métodos analíticos deterministas (es decir, que no usan la probabilidad). Sin embargo, el esfuerzo computacional requerido por estos métodos aumenta excesivamente conforme aumenta la complejidad del problema, incrementándose también el número de errores y por tanto se hacen inadecuados para la resolución de los más complejos. Este es el motivo principal por el que se están buscando métodos alternativos a los deterministas que resulten más eficientes y ofrezcan buenas soluciones. Estos nuevos métodos son algoritmos de optimización estocásticos (es decir, basados en la probabilidad), siendo mucho más eficientes que los deterministas. Estos métodos pueden ser heurísticos (métodos basados en el aprendizaje y la experiencia para resolver problemas),

metaheurísticos (métodos heurísticos que pueden ser empleados de forma general al asumir ciertas variables de los problemas), basados en poblaciones, entre otros [1]

En esta búsqueda de nuevos métodos de optimización, muchos investigadores han vuelto su vista a la naturaleza, ya que muchos de los procesos que ocurren en esta pueden ser vistos como la búsqueda de soluciones a distintos problemas, es decir procesos de optimización. Para cada problema que nos encontramos en la naturaleza, esta ha sido capaz de ofrecer soluciones eficaces, con poca o nula comprensión de este. Además, estos se resuelven a través de estrategias relativamente sencillas, por lo que son fáciles de implementar. Por ejemplo, podemos encontrar procesos de adaptación al medio (como estrategias de búsqueda de alimento o pareja), interacción con otras especies (como la competencia o la depredación, tanto a nivel individual como en grupos), estrategias a nivel de ecosistemas (como la colonización) o la evolución, que puede ser considerada como un proceso de optimización en sí misma. Debido a estas características muchos investigadores crearon nuevos métodos imitando la naturaleza, dando origen a los algoritmos bioinspirados. Al principio se usaron métodos específicos para problemas, pero más recientemente se han ido añadiendo métodos que funcionan en un amplio rango de problemas, pudiendo aplicarse de forma general.

En cuanto a sus procedimientos, estos algoritmos se centran en buscar la solución óptima a un problema, en el que todas sus posibles soluciones se pueden representar gráficamente como puntos en el espacio (la nube de soluciones del problema). Cada posición o punto en este espacio es una solución. Los algoritmos de optimización cuentan con una función que se encarga de comprobar lo buena que es la solución, lo que se ha llamado fitness de la solución. Las soluciones óptimas, es decir las que tienen mayor fitness, se encuentran dispersas aleatoriamente en la nube de soluciones, con una disposición en la que conforme más se va acercando el algoritmo a cualquiera de las soluciones óptimas, el fitness va a ir aumentando progresivamente hasta llegar a esta. Cada una de estas soluciones óptimas se conoce como **Óptimo local** (solución con más fitness en una región del espacio, por ejemplo, pueden ser mínimos o máximos del problema), de entre los cuales uno de ellos tendrá el mayor fitness de todas las soluciones, siendo el **Óptimo global** (por ejemplo, el máximo o el mínimo del problema). Debido a la característica azarosa de estos algoritmos, por ser estocásticos, es posible que en el proceso el algoritmo pierda algunos óptimos locales, e incluso el óptimo global, lo que llevaría al algoritmo a quedarse atascado en un óptimo local, por lo que no encontraría la mejor solución al problema. Para evitar este fenómeno, muchos algoritmos cuentan con procedimientos para asegurar que se encuentran el mayor número de óptimos posibles, dotando al algoritmo de lo que se conoce como **habilidad de búsqueda global**.

Espacio de búsqueda

Definición: en optimización un espacio de búsqueda es el dominio de la función a se busca optimizar, es decir el espacio de soluciones, donde se encuentran las soluciones candidatas.

Una búsqueda sobre un espacio consiste en generar una sucesión de puntos del espacio en el que cada punto se obtiene del anterior por medio de una serie de transformaciones o movimientos. En problema de optimización se realizan recorridos sobre el espacio de posibles soluciones y se selecciona la mejor solución encontrada en el recorrido.

Búsqueda local: una búsqueda local es un proceso que, dada la solución actual en la que se encuentra el recorrido, selecciona iterativamente una solución de su entorno cercano para continuar la búsqueda.

Búsqueda global: las búsquedas locales pueden quedar atrapada en óptimos locales, por lo que las búsquedas globales incorporan mecanismos para evitar esta situación, como son volver a comenzar la búsqueda desde otra solución inicial o permitir movimientos de empeoramiento de la solución actual, entre otros.

Si se conoce el espacio de búsqueda se puede seleccionar el tipo de algoritmo adecuado, sin embargo, esta situación raras veces se presenta por lo que se deben preferir métodos que realicen búsquedas globales. **Siempre que el costo computacional lo permita.**

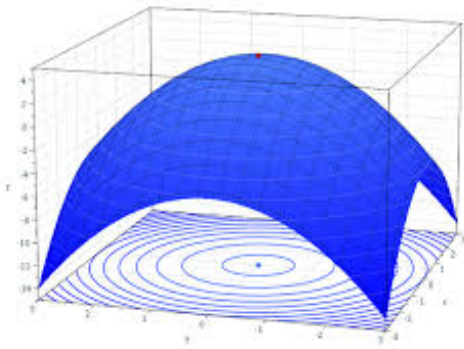


Figura 3. Espacio de búsqueda con máximo único

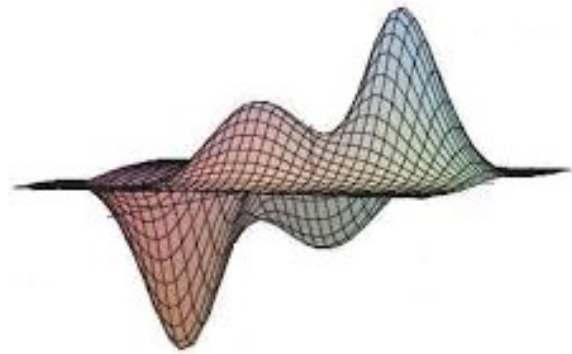


Figura 4. Espacio de búsqueda con múltiples óptimos locales

Antes de intentar resolver un problema o encontrar una solución a este, primero debemos identificar a que tipo de problema nos enfrentamos. El identificar la naturaleza de una problemática nos permite recrear una solución mucho más adecuada, satisfactoria, eficaz y de mayor calidad. Un esquema que se presenta para la identificación de los problemas tratados se denomina “caja negra para modelado de sistemas computacionales” [2]. Bajo la premisa anterior, podemos extraer tres características principales para la resolución de un problema computacional: los datos de entrada, el modelado o función que permitirá tratar o manipular los datos de entrada, y los datos de salida que son resultado del procesamiento de datos de entrada en un modelo, el desconocimiento de alguno de estos elementos nos permitirá conocer de que tipo de problema estaremos tratando. Estas tres clases de problemas son reconocidas como optimización, modelado y simulación.

Optimización

El modelo y la salida son conocidos, por lo que el objetivo es encontrar los valores de entrada que produzcan la mejor salida, por ejemplo, el problema de las 8 reinas, donde el modelo consiste en calcular si una reina choca con otra, y el resultado es tener 8 reinas en un tablero sin chocar, así pues, solo nos falta conocer sus posiciones, la entrada.

Modelado

Por otro lado, el modelo de un problema sucede cuando se conoce la entrada y la salida de un sistema, es el tipo de problema más común durante el aprendizaje de la programación, con ejemplos como concatenar dos cadenas, donde sabemos las cadenas que queremos concatenar, y que al final estarán juntas, por lo que solo resta encontrar la forma de procesarlas, modelando el problema.

Simulación

En cuanto a la simulación, se tiene la entrada y el modelo, y buscamos conocer la posible salida, se puede usar para predecir procesos reduciendo los costos que implicaría realizarlos realmente. Por ejemplo, la predicción meteorológica es un problema de simulación pues se conocen datos meteorológicos antiguos y un modelo usado para predecir como serán estos mismos datos dado cierto tiempo.

Ejemplo: un grupo de estudiantes tiene la tarea de construir un sistema robótico para jugar al tenis de mesa. Para cada una de las siguientes capacidades que debe exhibir el sistema, indique si se trata de un problema de optimización, modelado o simulación:

- a. Predecir dónde rebotará la pelota: **simulación**, ya que implica saber los datos entrada (posición, velocidad, fuerza, etc.) y simular el movimiento de la pelota usando algún modelo cinemático.
- b. Aprender el comportamiento del oponente: **modelado**. Implica que a partir del comportamiento de un individuo se cree un modelo, los datos tanto de entrada como salida son las acciones del individuo.
- c. Decidir dónde golpear la pelota a continuación para que el oponente tenga la menor posibilidad de devolverla: **optimización**, dado que se trata de minimizar la probabilidad de que golpee la pelota.

3. Algoritmos Genéticos (AG)

Un algoritmo genético es un método de resolución de problemas que utiliza la genética como modelo de resolución de problemas. Es una técnica de búsqueda para encontrar soluciones aproximadas a problemas de optimización y búsqueda.

- Son algoritmos estocásticos donde la aleatoriedad juega un papel importante. Tanto los procesos de selección y reproducción necesitan de funciones aleatorias.
- Siempre se considera una población de soluciones.
- Robustos pues tienen capacidad de desempeñarse bien de forma consistente en un amplio rango de problemas

La Figura 5 muestra un esquema general del funcionamiento de un algoritmo genético.

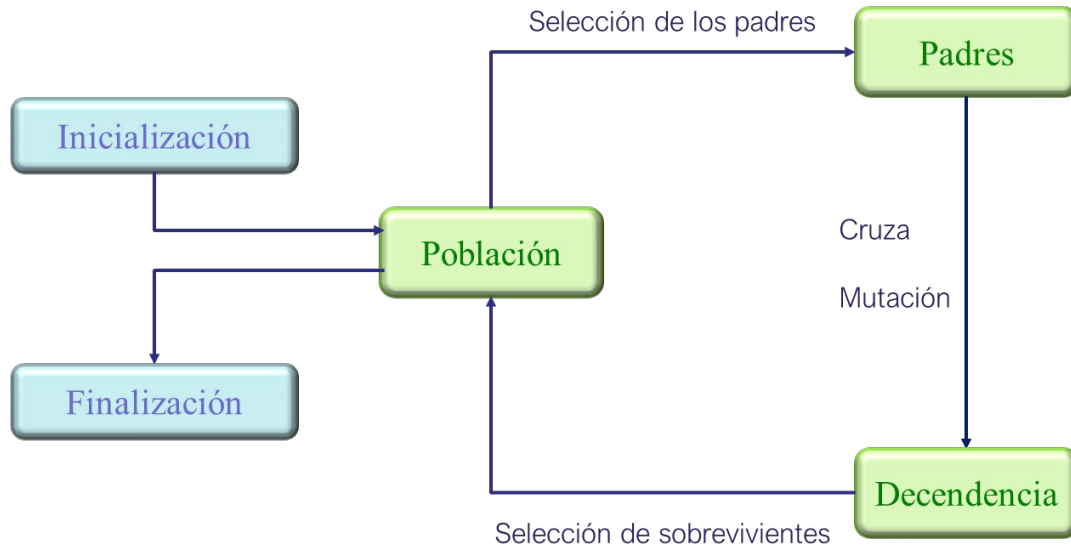


Figura 5. Esquema general de algoritmo genético

Los principales elementos del algoritmo genéticos son:

- Representación del cromosoma
- Función de evaluación.
- Población
- Mecanismo de selección de los padres
- Mecanismo de los operadores de cruce y mutación
- Mecanismo de reemplazo

Cromosoma

Un individuo o cromosoma es una sola solución a un problema dado. El primer paso para implementar un AG es la representación pues se deben especificar y almacenar las posibles soluciones de manera que puedan ser manipuladas por una computadora. Un individuo agrupa dos formas de soluciones como se indica a continuación:

- El cromosoma, que es la información "genética" sin procesar (genotipo) que trata el AG.
- El fenotipo, que es lo expresivo del cromosoma en los términos del modelo.

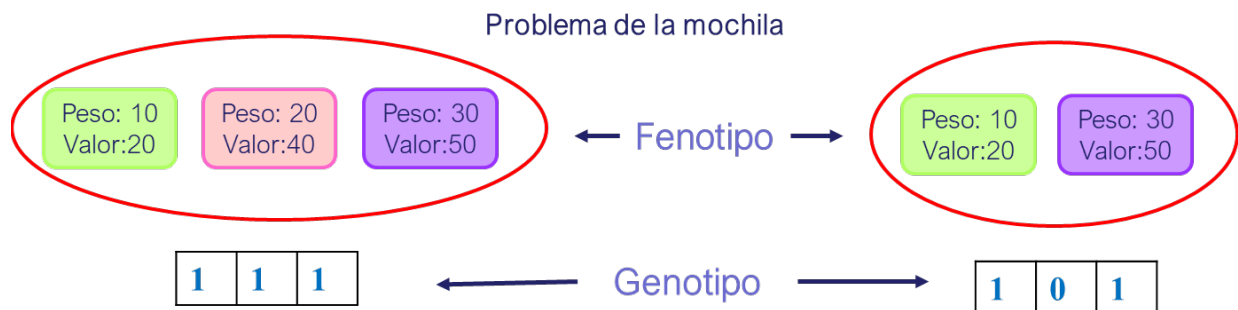


Figura 6. Ejemplo de codificación (Problema de la mochila)

Un gen es la unidad más básica en un AG. Los cromosomas se encuentran conformados por genes, donde cada gen representa una variable del problema.

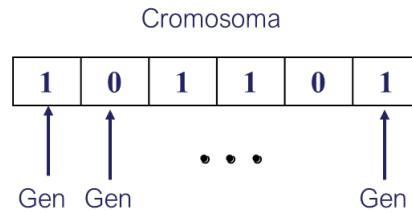


Figura 7. Ejemplo de cromosoma y gen

Existen varias formas de codificar los genes que componen un cromosoma, dependiendo del problema a resolver debe realizarse la selección de la codificación que mejor se adapte a dicho problema.

Codificación Binaria

Cromosoma 1	1100101110
Cromosoma 2	1001010111

Codificación Enteros

Cromosoma 1	182369547
Cromosoma 2	271935468

Figura 8. Ejemplo de codificación de cromosomas

Debe existir una relación o mapeo entre el genotipo y el fenotipo que permite convertir las soluciones del modelo en un formato con el que el AG pueda trabajar, así como el proceso inverso.

$$\begin{array}{|c|c|c|c|c|c|} \hline 1 & 0 & 1 & 1 & 0 & 1 \\ \hline \end{array} = 45$$

$$1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$$

$$= 32 + 8 + 4 + 1 = 45$$

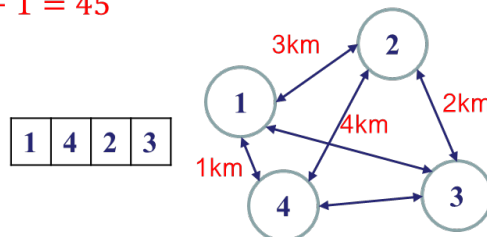


Figura 9. Ejemplo de función de mapeo genotipo/fenotipo

Función de aptitud

La aptitud de un individuo en un algoritmo genético mide que tan buena o mala es la solución que se encuentra codificada en dicho individuo. Generalmente es el valor evaluado en la función objetivo del individuo.

La Figura 10 muestra un ejemplo donde se requiere minimizar la función dada. Desde este punto de vista el individuo 1 tiene el mejor valor de aptitud o *fitness*.

$$f(x) = ABS \left| \frac{x - 5}{2 + Sen(x)} \right|$$

	Genotipo	Fenotipo	$f(x)$
Individuo 1	0011	3	0.93
Individuo 2	1111	15	3.77
Individuo 3	0001	1	1.40
Individuo 4	1000	8	1

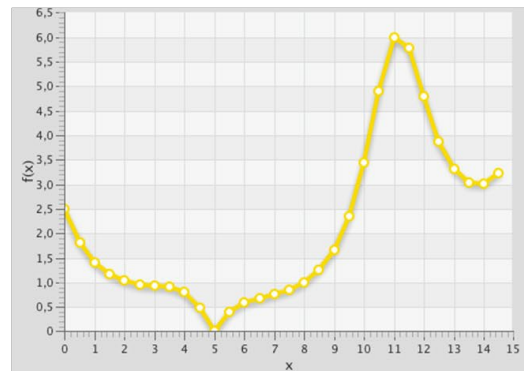


Figura 10. Ejemplo de función objetivo

La función de aptitud cumple varias tareas:

- Representa los requisitos a los que la población debe adaptarse.
- Constituye la base para la selección y, por lo tanto, facilita las mejoras.
- Una función o procedimiento que asigna una medida de calidad a los genotipos.

Población

Es un multiconjunto de individuos, en donde los individuos son objetos estáticos que no cambian ni se adaptan; es la población la que lo hace. La población forma la unidad de evolución.

- Generalmente, la población inicial se genera de forma aleatoria.
- Alguna heurística podría ser usada para la población inicial lo que permitiría una convergencia más rápida, pero se corre el riesgo de falta de diversidad que no permita explorar todo el espacio de soluciones.
- Entre más grande el tamaño de la población más fácil explorar el espacio de soluciones.
- De acuerdo con [3], el tiempo requerido por un AG converge a $O(n \log n)$ donde n es el tamaño de la población. Por lo que una población grande puede ser útil, pero requiere mayor costo computacional.
- Una población inicial de 100 individuos es común, pero depende de los recursos computacionales disponibles
- Generalmente el tamaño de la población es una constante y no cambia durante el proceso evolutivo.

Selección de padres

Es el proceso de elegir dos padres de la población para cruzar y crear nuevos individuos. La presión de selección se define como el grado al que se favorece los mejores individuos para ser seleccionados como padres. Por lo general, podemos distinguir dos tipos de esquema de selección:

- Selección proporcional.
- Selección basada en ordinales.

Selección por ruleta

La selección de ruleta es una de las técnicas tradicionales de selección de AG. Es un operador de selección **proporcional** donde se selecciona un individuo con una probabilidad proporcional a su aptitud. Se llama de la ruleta porque imita el proceso de girar una ruleta y seleccionar un punto al azar en esta.

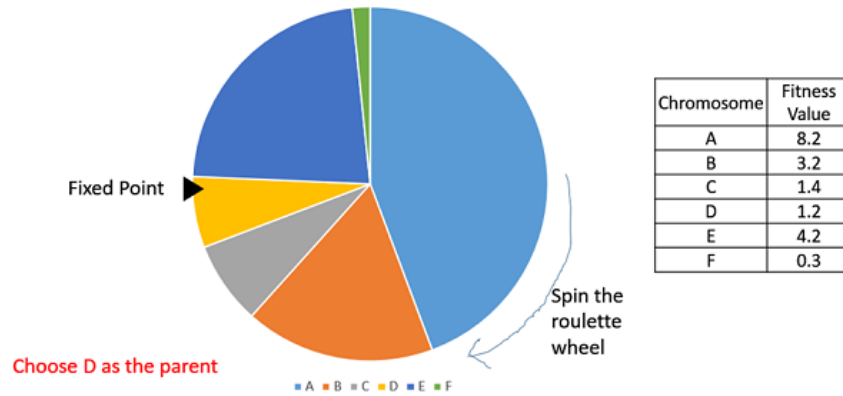


Figura 11. Método de la ruleta

Ejemplo del método de la ruleta:

$$f(x) = 4x_1 + 5x_2 + 6x_3 + 3x_4$$

	x_1	x_2	x_3	x_4	$f(x)$	P_{sel}	P_{sel_acum}
Individuo 1	0	1	1	0	11	0.29730	0.29730
Individuo 2	1	0	1	0	10	0.27027	0.56757
Individuo 3	0	1	0	0	5	0.13514	0.70270
Individuo 4	0	1	1	0	11	0.29730	1.00000

$$P_{sel} = \frac{f(x)}{Total}$$

Total: 37

Suma total de las aptitudes de todos los individuos en la población

$$\text{Individuo 1} \rightarrow P_{sel} = \frac{11}{37} = 0.29730$$

$$\text{Individuo 2} \rightarrow P_{sel} = \frac{10}{37} = 0.27027$$

$$\text{Individuo 3} \rightarrow P_{sel} = \frac{5}{37} = 0.13514$$

La P_{sel_acum} es la suma acumulada de P_{sel} de cada individuo.

El primer paso es realizar la suma total de las aptitudes de todos los individuos de la población, luego se calcula la probabilidad proporcional (columna P_{sel}) de selección de cada individuo mediante la razón entre la aptitud del individuo y la suma total de las aptitudes de la población. Finalmente debe

realizarse la suma acumuladas de las probabilidades de selección de todos los individuos (columna P_{sel_acum}).

	x_1	x_2	x_3	x_4	$f(x)$	P_{sel}	P_{sel_acum}
Individuo 1	0	1	1	0	11	0.29730	0.29730
Individuo 2	1	0	1	0	10	0.27027	0.56757
Individuo 3	0	1	0	0	5	0.13514	0.70270
Individuo 4	0	1	1	0	11	0.29730	1.00000

Total: 37

Valor aleatorio 1: 0.4771

Padre 1 → Individuo 2

Valor aleatorio 2: 0.9341

Padre 2 → Individuo 4

Buscar en P_{sel_acum} el valor igual o mayor al número aleatorio generado

Finalmente debe generarse un número aleatorio para cada individuo a seleccionar, donde se debe comparar el valor aleatorio con la P_{sel_acum} y el individuo para el cual P_{sel_acum} es igual o mayor a ese número aleatorio es seleccionado como padre.

Las características más relevantes de este método son:

- Es sencillo de implementar.
- Los individuos con más aptitud tienden a dominar la población en pocas generaciones
- La tasa de evolución depende de la varianza de la aptitud en la población.
- Un individuo apto contribuirá más al valor objetivo, pero si no lo supera, el siguiente cromosoma en la línea tiene una oportunidad, aunque tenga un valor de aptitud bajo.

Selección por ranking

La selección de ranking ordena la población por su aptitud y cada individuo recibe una probabilidad de selección de acuerdo con su posición. El peor individuo tiene ranking 0, mientras que el mejor individuo tendrá ranking de $N-1$ donde N es el número de individuos en la población. El mapeo del ranking a la probabilidad de selección se puede calcular de muchas maneras, por ejemplo, lineal o exponencialmente. La selección por ranking es útil cuando el valor de aptitud de cada individuo es muy similar. Si en esta situación (ver Figura 12) aplicáramos la selección de ruleta no existiría presión de selección.

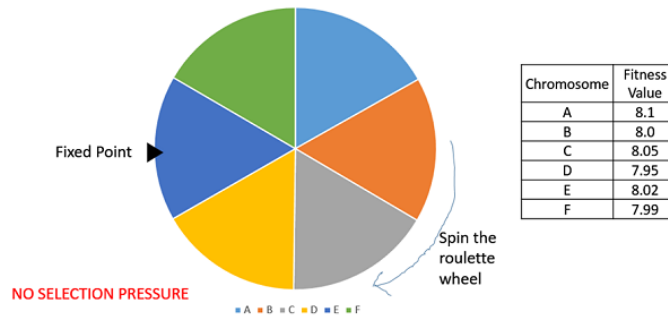


Figura 12. Ejemplo de valores de aptitud sin presión de selección

La probabilidad de selección lineal para cada individuo se calcula de acuerdo con la siguiente expresión:

$$P_{s_rank}(i) = \frac{(2 - s)}{N} + \frac{2i(s - 1)}{N(N - 1)}$$

donde $1 < s \leq 2$ y N es el número de individuos en la población.

Ejemplo: la Tabla 1 muestra el cálculo de la probabilidad de selecciones de los mismos individuos, pero con diferentes valores de S .

Tabla 1. Ejemplo de probabilidad de selección con método de ranking

	Aptitud $f(x)$	Ranking	P_{s_rank} ($s = 2$)	P_{s_rank} ($s = 1.5$)
Individuo 1	1	0	0	0.125
Individuo 3	2	1	0.17	0.208
Individuo 4	3	2	0.33	0.292
Individuo 2	5	3	0.50	0.375

Una vez calculada la probabilidad de selección (P_{s_rank}), se sigue el mismo procedimiento de selección que en la ruleta con generación de valores aleatorios para la selección de los padres, es decir la diferencia de los métodos consiste en cómo se calcula la probabilidad de selección.

	Aptitud $f(x)$	Ranking	P_{s_rank} ($s = 2$)
Individuo 1	1	0	0
Individuo 3	2	1	0.17
Individuo 4	3	2	0.33
Individuo 2	5	3	0.50

Valor aleatorio 1: 0.47

Padre 1 → Individuo 2

Valor aleatorio 2: 0.15

Padre 2 → Individuo 3

Buscar en P_{s_rank} el valor igual o mayor al número aleatorio generado

Figura 13. Ejemplo de selección con método de ranking

Selección por torneo

En este método Se seleccionan k individuos al azar y de estos se selecciona el que tenga mejor valor de aptitud para ser un padre. Se repite el proceso para seleccionar el otro padre.

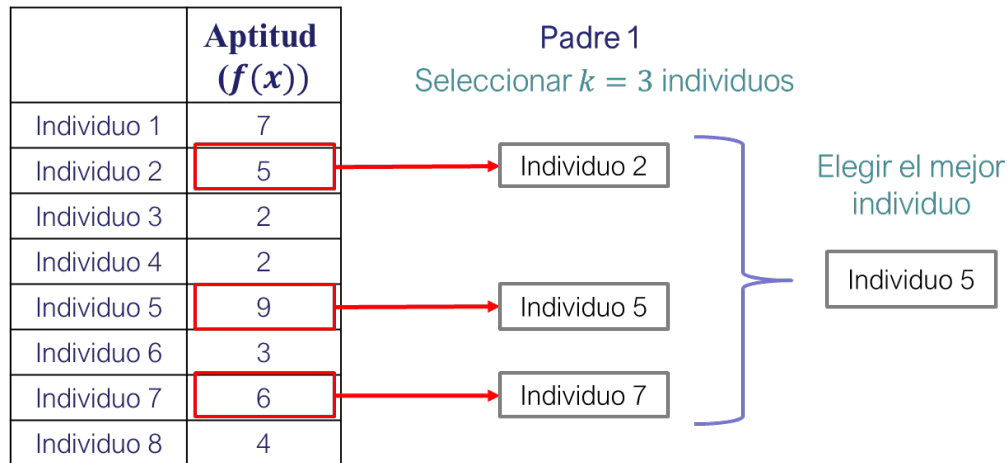


Figura 14. Método de selección por torneo

Cruza

La recombinación o cruce es un operador esencial para la evolución natural. Toma dos individuos de la población (padres) y produce nuevos individuos mezclando los genes que se encuentran en los originales. Lo que se espera es que la descendencia, simplemente combinando todas las buenas características de sus padres, puede superar a sus antepasados.

Probabilidad de Cruza

La probabilidad de cruce (P_c) es un parámetro básico de los AG que indica si todos los individuos seleccionados como padres se reproducen.

- $P_c = 1$, todos los individuos seleccionados se cruzan, no hay supervivientes de la generación anterior.
- $P_c = 0$, ningún individuo se cruza. La nueva generación está hecha de copias exactas de los cromosomas de la población anterior.
- La cruce se realiza con la esperanza de que los nuevos cromosomas sean mejores. Sin embargo, es bueno dejar que una parte de la población antigua sobreviva a la próxima generación.
- Estudios empíricos demuestran que una probabilidad de cruce entre 0,65 y 0,85 logra buenos resultados. Esto significa que un cromosoma tiene una probabilidad entre 0.15 y 0.35 de sobrevivir a la siguiente generación.

Para la implementación de esto se siguen los siguientes pasos:

- a. Generar un número aleatorio entre 0 y 1.
- b. Si el número generado es menor que P_c se realiza la cruce.

- c. En caso contrario, los individuos seleccionados como padres pasan sin modificación a la siguiente generación (supervivientes).

La Figura 15 muestra un ejemplo donde si se realiza cruce, ya que el valor del número aleatorio generado es menor que la probabilidad de cruce. La Figura 16 muestra un ejemplo donde no se realiza cruce pues el valor del número aleatorio generado es mayor que la probabilidad de cruce, en este caso los individuos que fueron seleccionados como padres pasan como sobrevivientes a la siguiente generación.

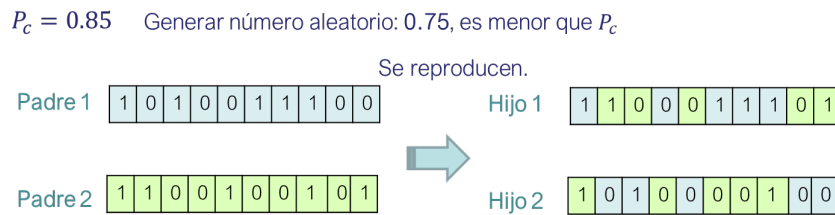


Figura 15. Ejemplo en que se realiza cruce.

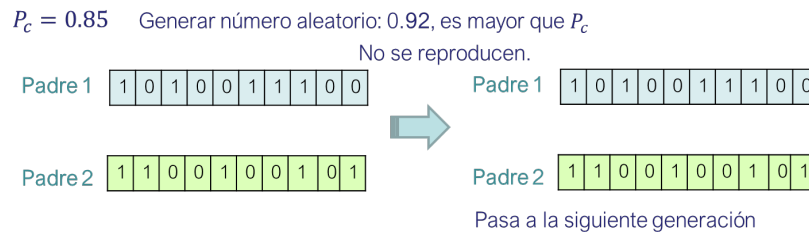


Figura 16. Ejemplo donde no se realiza cruce

Cruza de 1 punto

Pasos:

- Selección de 2 individuos para ser padres.
- Se elige un punto de cruce al azar (posición de un gen dentro del individuo).
- Finalmente, los genes de los 2 individuos se intercambian a partir del punto de cruce

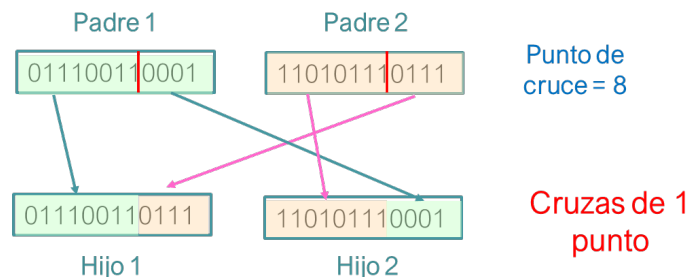


Figura 17. Cruza de 1 punto

Cruza de 2 puntos

Pasos:

- Selección de 2 individuos para ser padres.

- ii. Se elige dos puntos de cruce al azar (posición de un gen dentro del individuo).
- iii. Finalmente, los genes de los 2 individuos se intercambian a partir de los puntos de cruce.

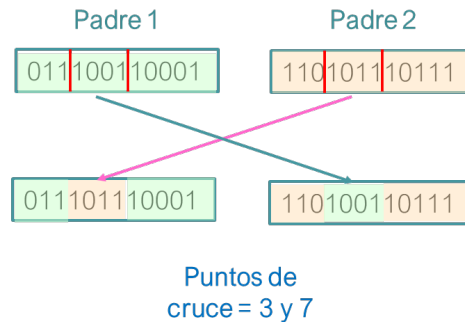


Figura 18. Cruza de 2 puntos

Cruza uniforme

Tratar cada gen de forma independiente y elegir aleatoriamente de qué padre se debe heredar cada gen. Esto se implementa generando una variable aleatoria del tamaño de cromosoma a partir de una distribución uniforme $[0, 1]$. Si el valor es menor que un umbral ($u=0.5$), se heredan los genes del primer padre, de otra forma se heredan del segundo. La selección de genes se invierten en el segundo hijo, es decir que si el valor es menor que un umbral ($u=0.5$), se heredan los genes del segundo padre, de otra forma se heredan del primer padre.



Figura 19. Cruza uniforme

Mutación

Consiste en cambiar el valor de los genes. En la evolución natural generalmente produce individuos no aptos. Sin embargo, es la optimización, algunos cambios aleatorios pueden ser una buena manera de explorar el espacio de búsqueda rápidamente.

- Evita quedar atrapados en mínimos o máximos locales.
- Si la cruce permite explotar la solución actual para encontrar otras mejores, la mutación ayudará a explorar todo el espacio de búsqueda.
- La mutación puede verse como un operador de fondo para mantener la diversidad genética en la población al introducir nuevo material genético.

Probabilidad de mutación

La probabilidad de mutación (P_m) es otro parámetro básico de los AG que indica que parte del cromosoma debe ser mutado.

- $P_m = 1$, todos los genes del individuo deben ser mutados.
- $P_m = 0$, ningún gen del individuo debe ser mutado.
- La mutación no debería ocurrir muy a menudo, porque entonces el AG se transforma en una búsqueda aleatoria, por lo que se eligen valores bajos de P_m .

Para la implementación de esto se siguen los siguientes pasos:

- Generar un número aleatorio entre 0 y 1 por cada gen del individuo.
- Si el número correspondiente a un gen es menor que P_m se muta el gen.
- En caso contrario, el gen no se modifica.

$P_m = 0.1$	Hijo1	1	0	1	0	0	1	1	1	0	0
		0.25	0.32	0.17	0.02	0.65	0.92	0.08	0.16	0.29	0.78
		1	0	1	1	0	1	0	1	0	0
	Hijo2	1	1	0	0	1	0	0	1	0	1
		0.65	0.24	0.06	0.18	0.68	0.71	0.63	0.27	0.17	0.19
		1	1	1	0	1	0	0	1	0	1

Figura 20. Ejemplo de mutación

La mutación se produce después del proceso de cruce y los 2 hijos resultantes de dicho proceso deben ser mutados. La Figura 20 da un ejemplo de esto. Se puede observar que solo las posiciones donde la probabilidad es menor a 0.1 se cambian, en el hijo 1 las posiciones 4 y 7, mientras que en el hijo 2 solo la posición 3 es cambiada.

Mutación por inversión (flip) de bits

- Generar un cromosoma de mutación con valores aleatorios de 0 y 1 (distribución uniforme) de la misma dimensión que el cromosoma.
- Las posiciones que correspondan con valores menores a la probabilidad de mutación son invertidas.

Individuo sin mutar	1	0	1	0	0	1	1	1	0	0
Cromosoma de mutación	0.7	0.02	0.4	0.9	0.08	0.3	0.7	0.04	0.8	0.06
Individuo mutado	1	1	1	0	1	1	1	0	0	1

Figura 21. Mutación por inversión de bits

La Figura 21 muestra un ejemplo de mutación por inversión de bits. Primero el cromosoma de mutación indica que posición debe ser mutada y luego el valor del gen en esa posición se invierte. Si el valor era 1 se cambia a 0, si era 0 se cambia a 1.

Mutación por intercambio

- Seleccionar 2 posiciones al azar
- Intercambiar los bits correspondientes a esas posiciones.

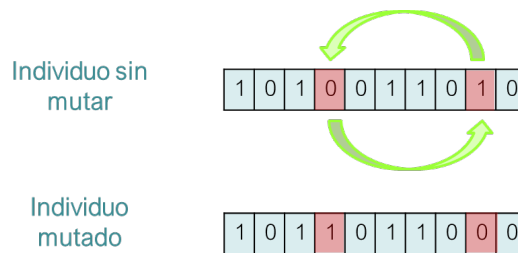


Figura 22. Mutación por intercambio

Mutación aleatoria

- Generar un cromosoma de mutación con valores aleatorios de 0 y 1 (distribución uniforme) de la misma dimensión que el cromosoma.
- Las posiciones que correspondan con valores menores a la probabilidad de mutación se cambian por algún valor del grupo de valores permitidos

Valores permitidos: {0 – 20}

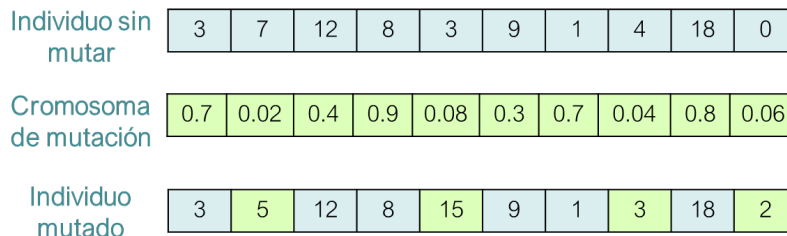


Figura 23. Mutación aleatoria

Mutación uniforme

- Generar un cromosoma de mutación con valores aleatorios de 0 y 1 (distribución uniforme) de la misma dimensión que el cromosoma.
- Las posiciones que correspondan con valores menores a la probabilidad de mutación se cambian por algún valor de un rango continuo de valores permitidos.

$$\langle x_1, \dots, x_n \rangle \rightarrow \langle x'_1, \dots, x'_n \rangle$$

donde $x_i, x'_i \in [L_i, U_i]$, L_i es el mínimo valor permitido para la i -ésima variable y U_i es el máximo valor permitido para la i -ésima variable

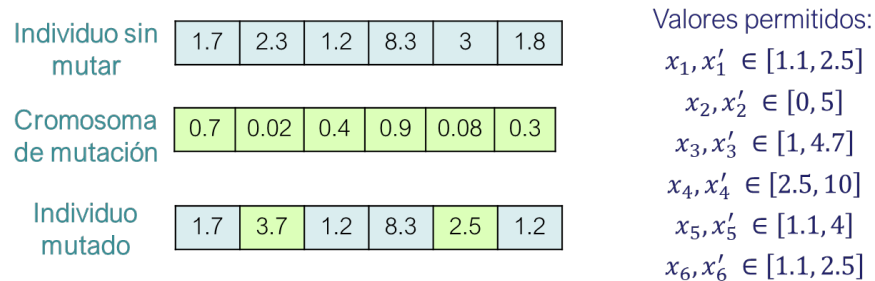


Figura 24. Mutación uniforme

Reemplazo (selección de supervivientes)

Es el último paso del ciclo de reproducción. Es el proceso de reducir un conjunto de P padres y L hijos a un conjunto de P individuos que conforman la siguiente generación. Una vez que se producen los descendientes, un método debe determinar cuál de los miembros actuales de la población, si alguno, debe ser reemplazado por las nuevas soluciones.

Reemplazo de ambos padres

Consiste en que la descendencia producida siempre reemplaza a sus padres, como se muestra en la Figura

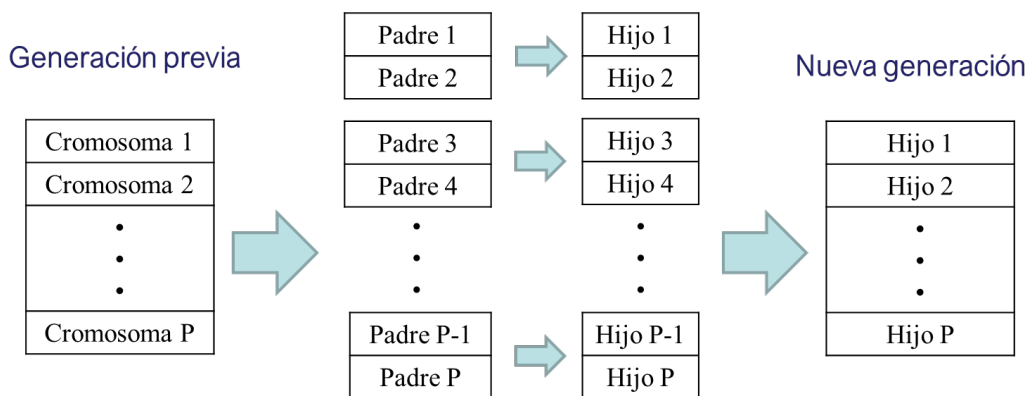


Figura 25. Reemplazo de ambos padres

Reemplazo padre más débil

En este caso solamente se reemplazan los padres si el fitness de los hijos es mejor, en caso contrario los padres sobreviven. Es decir que, si la descendencia producida no es mejor que los padres, entonces no sobreviven a la siguiente generación.

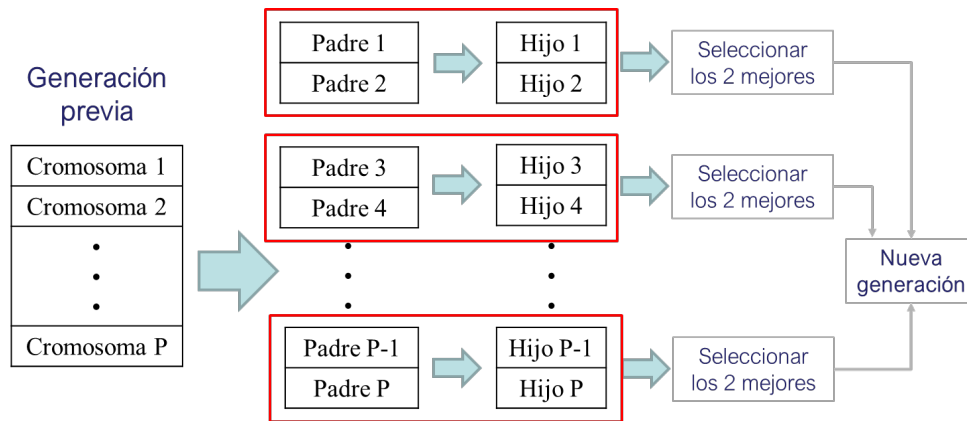


Figura 26. Reemplazo del padre más débil

Criterios de finalización

Generaciones máximas: el algoritmo genético se detiene cuando ha evolucionado el número especificado de generaciones.

Tiempo transcurrido: el proceso genético finalizará cuando haya transcurrido un tiempo específico.

Sin cambios en la aptitud: el proceso genético terminará si no hay cambios en el mejor valor de aptitud de la población para un número específico de generaciones.

4. Programación Genética

La programación genética (PG) es una técnica de computación evolutiva que resuelve problemas automáticamente sin requerir que el usuario conozca o especifique la forma o estructura de la solución de antemano [4]. La programación genética logra este objetivo de programación automática (a veces también llamada síntesis de programa o inducción de programa) mediante la reproducción genética de una población de programas de computadora utilizando los principios de la selección natural darwiniana y operaciones inspiradas biológicamente [5].

En el nivel más abstracto, GP es un método sistemático e independiente del área de conocimiento para hacer que las computadoras resuelvan problemas automáticamente a partir de una declaración de alto nivel de lo que se debe hacer [4]. La solución encontrada por PG es un programa que resuelve muchas instancias de problemas.

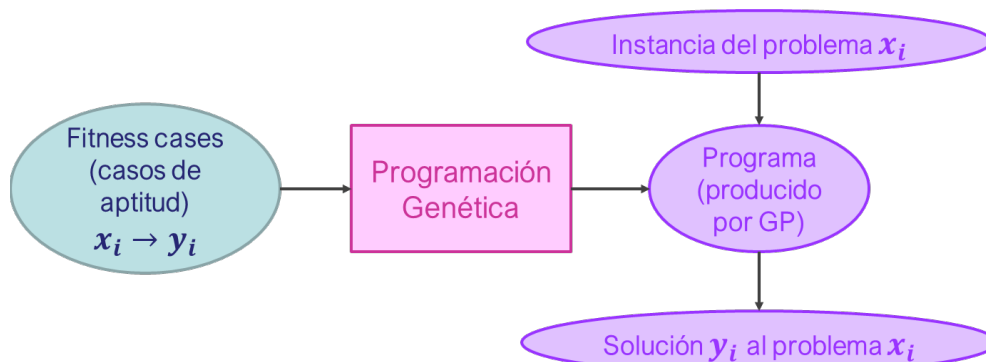


Figura 27. Esquema general de PG

Ejemplo: regresión simbólica, en el cual, dados los valores de entrada y salida, PG tiene que aproximar la función que los genero, es decir que dado un conjunto de valores de la forma (x, y) , encontrar la función que los genera.

x	-7	-5	-3	0	1	3	5	7
y	-2	-0.2	0.2	0	-0.1	-0.3	0.3	1.5

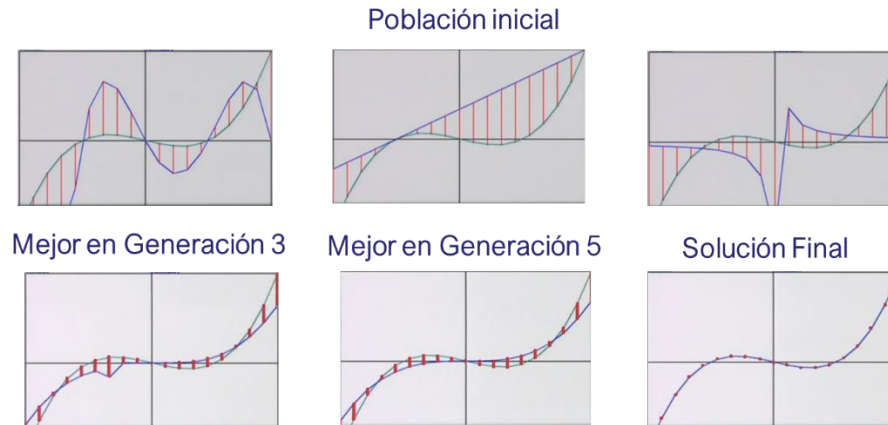


Figura 28. Ejemplo de PG para regresión simbólica

Los árboles sintácticos usados en PG como cromosomas capturan expresiones en una sintaxis formal dada. Según el problema en cuestión, esto puede ser la sintaxis de expresiones aritméticas, expresiones lógicas o código escrito en un lenguaje de programación. Al comparar con AG, se encuentran las siguientes diferencias más relevantes:

Representación: en lugar de una cadena de valores (AG), el individuo se representa como un árbol sintáctico.

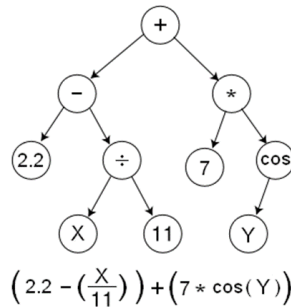


Figura 29. Individuo en PG

Activo vs. pasivo: PG generalmente desarrolla programas, las soluciones se pueden ejecutar sin procesamiento posterior (estructuras activas), mientras que los GA generalmente operan en cadenas binarias codificadas (estructuras pasivas), que requieren procesamiento posterior.

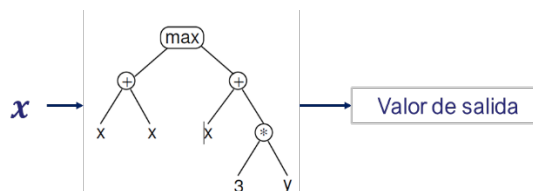


Figura 30. Estructura activa en PG

Tamaño fijo vs. variable: en los AG, la longitud de la cadena que representa un individuo es fija y se define antes de que comience el procedimiento de solución. Sin embargo, un árbol en PG puede variar en longitud a lo largo de la ejecución.

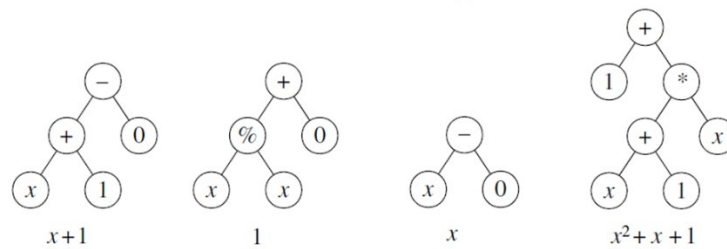


Figura 31. Representaciones variables en PG

La Figura 32 muestra el diagrama de flujo de la PG.

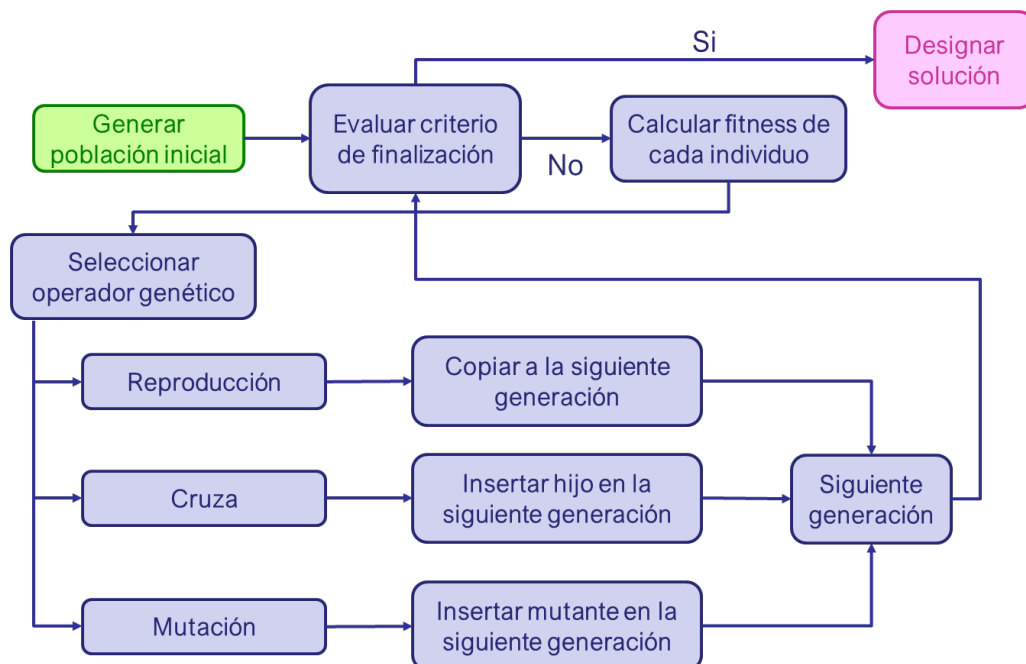


Figura 32. Diagrama de flujo de PG

Pasos preparatorios

El usuario humano comunica la descripción de alto nivel del problema al sistema de programación genética realizando ciertos pasos preparatorios bien definidos. Estos 5 pasos preparatorios requieren que el humano defina:

- Conjunto de nodos terminales
- Conjunto de nodos función
- Medida de aptitud
- Parámetros de control
- Criterio de terminación

Representación

En PG, los programas generalmente se expresan como árboles de sintaxis. Durante la inicialización se requiere definir una profundidad máxima inicial (D_{max}) para los árboles y así evitar el fenómeno de *bloating*, el cual se da cuando se crean árboles demasiado complejos. También se deben definir un conjunto de funciones y un conjunto de terminales válidos.

El conjunto de estructuras posibles en PG es el conjunto de todas las composiciones posibles de funciones que se pueden realizar recursivamente a partir del conjunto $F = \{f_1, f_2, \dots, f_{N_{func}}\}$ de funciones y el conjunto $T = \{t_1, t_2, \dots, t_{N_{term}}\}$ de terminales. Cada función particular f_i toma un número específico $z(f_i)$ de argumentos lo que significa que la función f_i tiene aridad $z(f_i)$.

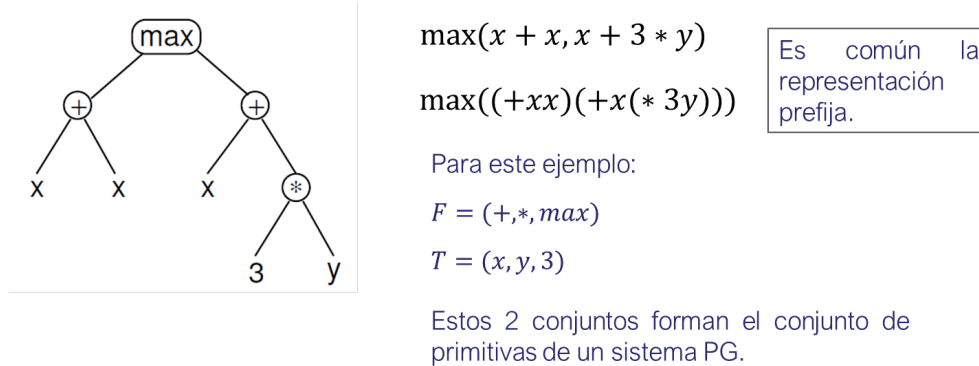


Figura 33. Ejemplo de representación en PG

El conjunto de las funciones puede incluir:

- Operaciones aritméticas (+, -, *, etc.),
- Funciones matemáticas (como sin, cos, exp y log),
- Operaciones booleanas (como AND, OR, NOT),
- Operadores condicionales (como If-Then-Else),
- Funciones que provocan la iteración (como Do-Until),
- Funciones que causan recursividad
- Cualquier otra función específica del dominio que pueda definirse.

Los terminales suelen ser variables (que pueden representar las entradas, sensores, detectores o variables de estado de algún sistema) o constantes (un número o la constante booleana NIL). Ocasionalmente, los terminales son funciones que no toman argumentos explícitos, la funcionalidad real de tales funciones radica en sus efectos secundarios sobre el estado del sistema (ejemplo la función *random*).

Función de aptitud

La tarea de medir que programas son buenos dentro de nuestro espacio de búsqueda es responsabilidad de la función de aptitud. La función de aptitud en GP es diferente de la usada en otros enfoques genéticos, dado que los individuos que evolucionan son programas, por lo que, para calcularla, generalmente, se requiere ejecutar todos los programas de la población.

Para esto se acostumbra a usar intérpretes y no compiladores, dado el costo de construir un compilador. Interpretar un árbol (programa) significa ejecutar los nodos en el árbol en un orden que garantice que los nodos no se ejecuten antes de que se conozca el valor de sus argumentos (si los hay). Esto generalmente se hace recorriendo el árbol recursivamente comenzando desde el nodo raíz y posponiendo la evaluación de cada nodo hasta que se conozcan los valores de sus hijos (argumentos).

Aptitud bruta (raw fitness): es la medida de la aptitud que se establece en la terminología natural del problema mismo. A menudo, la aptitud bruta se calcula sobre un conjunto de casos de aptitud. Un caso de aptitud corresponde a una situación representativa en la que se puede evaluar la capacidad de un programa para resolver un problema.

Casos de aptitud						Función
x	1	3	6	8	10	$f(x) = x^2 + 1$
y	2	10	37	65	101	

Figura 34. Aptitud bruta para PG

La aptitud bruta se calcula como la suma de las diferencias entre el valor esperado, especificado en los fitness cases, y los valores obtenidos por el individuo evaluado.

Parámetros de control

El paradigma de programación genética está controlado parámetros numéricos y cualitativas que seleccionan entre varias formas alternativas de ejecutar el paradigma.

- Tamaño de la población, valor sugerido $M = 500$
- Número de generaciones, valor sugerido $G = 51$. 1 generación inicial llamada generación 0 y 50 generaciones subsecuentes.
- Probabilidad de cruza, valor sugerido $P_c = 0.9$.
- Probabilidad de mutación, valor sugerido $P_m = 0$.
- Probabilidad de reproducción, valor sugerido $P_r = 0.1$.
- Profundidad máxima, valor sugerido $D_{max} = 6$.
- Profundidad máxima creación, valor sugerido $D_{create} = 17$

Población inicial

La inicialización de la población es el primer paso del proceso de evolución. Implica la creación de las estructuras del programa que luego evolucionarán. Generalmente se inicia con un grupo de miles de programas de computadora generados aleatoriamente.

Se inicia por seleccionar una función del conjunto F al azar para que sea la raíz del árbol. Se restringe la selección de la raíz del árbol al conjunto de funciones F porque se debe generar una estructura jerárquica y no una estructura con un solo nodo terminal.

Método “Full”

- Se selecciona una función de forma aleatoria con probabilidad uniforme del conjunto de funciones F para que sea la raíz del árbol; sea z la aridad de la función seleccionada.
- Se seleccionan n nodos con probabilidad uniforme del conjunto de funciones F , para que sean sus hijos, donde n debe corresponder a la aridad de la función en el nodo padre;
- Para cada función dentro de estos n nodos, se invoca recursivamente el método de crecimiento, es decir, sus hijos se seleccionan del conjunto F , a menos que el nodo tenga una profundidad igual a $d_{max} - 1$. En este último caso, sus hijos se seleccionan de T .

La profundidad de un nodo es el número de aristas que deben atravesarse para llegar al nodo a partir del nodo raíz del árbol. La profundidad de un árbol es la profundidad de su hoja más profunda. Este método garantiza que todas las ramas del árbol tienen la misma profundidad.

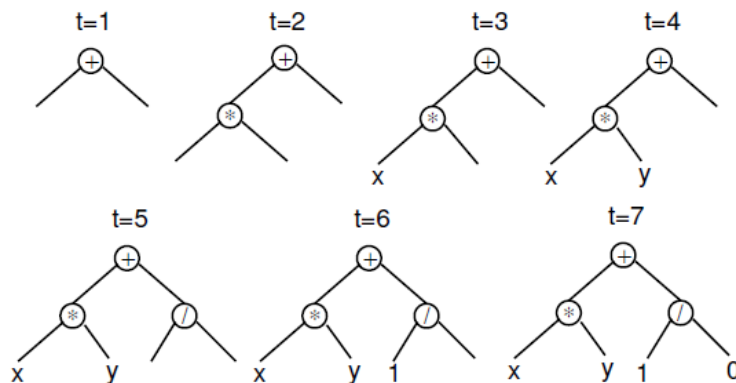


Figura 35. Ejemplo de creación de árbol con método full

Método “Grow”

- Se selecciona una función de forma aleatoria con probabilidad uniforme del conjunto de funciones F para que sea la raíz del árbol; sea z la aridad de la función seleccionada.
- Se seleccionan n nodos con probabilidad uniforme de la unión del conjunto de funciones y el conjunto terminal, $F \cup T$, para que sean sus hijos;
- Para cada función dentro de estos n nodos, se invoca recursivamente el método de crecimiento, es decir, sus hijos se seleccionan del conjunto $F \cup T$, a menos que el nodo tenga una profundidad igual a $d_{max} - 1$. En este último caso, sus hijos se seleccionan de T . Para los nodos terminales se detiene el crecimiento.

El rango de tamaños y formas de los programas producidos por el método **Full** puede ser limitado. El método **Grow**, por el contrario, permite la creación de árboles de formas y tamaños más variados.

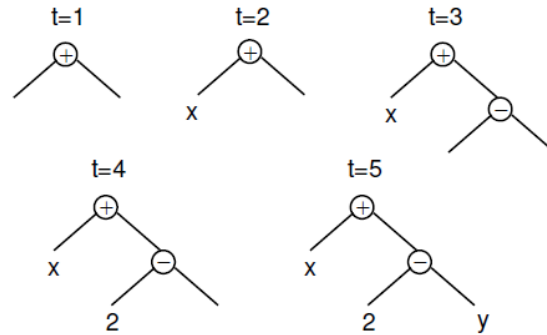


Figura 36. Ejemplo de creación de árbol con método grow

Método “*ramped half-and-half*”

Según Koza [4], las poblaciones generadas con cualquiera de los 2 métodos presentados anteriormente pueden mostrar falta de diversidad, por lo que propuso el método ***ramped half-and-half***. En este método:

- i. Una fracción $\frac{1}{d_{max}}$ de la población se inicializa con árboles de profundidad 1, otra fracción $\frac{1}{d_{max}}$ se inicializa con árboles de profundidad 2, y así consecutivamente hasta llegar a d_{max} .
- ii. Además, por cada grupo de las diferentes profundidades, la mitad de los árboles son inicializados con el método **Grow** y la otra mitad con el método **Full**.

Selección de padres

Al igual que con la mayoría de los algoritmos evolutivos, los operadores genéticos en PG se aplican a individuos que se seleccionan probabilísticamente en función de la aptitud. El autor de este enfoque [4], sugiere el uso del método de la ruleta, que ya fue explicado en la sección de AG.

Operador de reproducción

La operación de reproducción es asexual ya que opera con un solo padre y produce un solo hijo.

- i. Seleccionar un padre de la población de acuerdo con algún método de selección basado en la aptitud.
- ii. El individuo seleccionado es copiado, sin alteración, de la población actual a la nueva población.

Un individuo seleccionado como padre permanece en la población mientras se realiza la selección durante la generación actual. Es decir, la selección se realiza con reemplazo (reselección). Los padres pueden seleccionarse y, en general, se seleccionan más de una vez para la reproducción durante la generación actual. Esto se cumple para todos los operadores genéticos

Cruza

La operación de cruce (recombinación sexual) para PG crea una variación en la población al producir nuevos hijos creados con partes tomadas de cada padre

Cruza de subárboles

- Seleccionar 2 padre de la población de acuerdo con algún método de selección basado en la aptitud.
- Escoger 1 nodo al azar en cada árbol, a los cuales se les llama punto de cruce.
- Se produce 1 hijo eliminando el fragmento de cruce del padre 1 e insertando el fragmento de cruce en el padre 2 en el punto de cruce del padre 2. El segundo hijo se produce de forma simétrica.

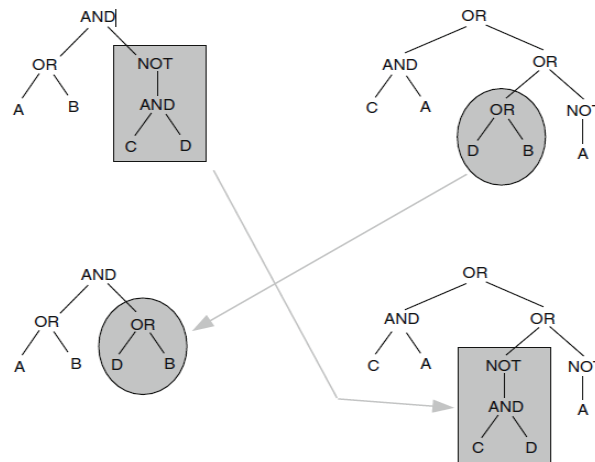


Figura 37. Ejemplo de cruce de subárbol

Cuando el punto de cruce de un padre es una hoja mientras y en el otro padre es el nodo raíz, la descendencia tendrá más profundidad que los padres. Si bien esto puede ser deseable en las primeras etapas de una corrida, en la presencia del fenómeno **bloat** puede ser necesario imponer algún límite al tamaño de la descendencia. Podemos ver en la Figura 38 que, con una profundidad máxima de 17, se pueden producir árboles con 131,072 nodos.

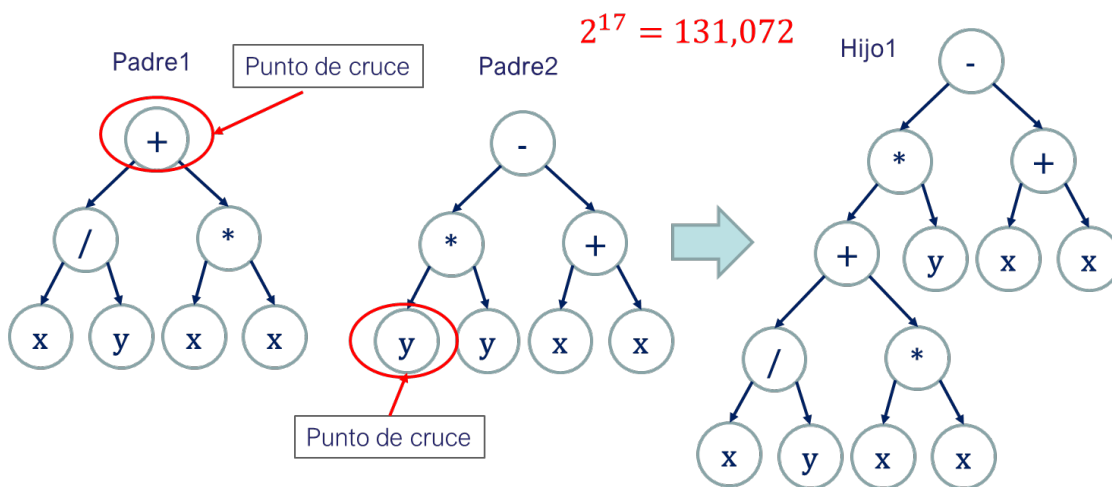


Figura 38. Cruza con fenómeno de bloating

En cambio, Si los puntos de cruce en cada padre se encuentran en nodos terminales, la operación de cruce simplemente intercambia terminales de un árbol a otro. El efecto de cruce, en este caso, es similar a una mutación puntual. Por lo tanto, la mutación puntual ocasionalmente es una parte de la operación de cruce.

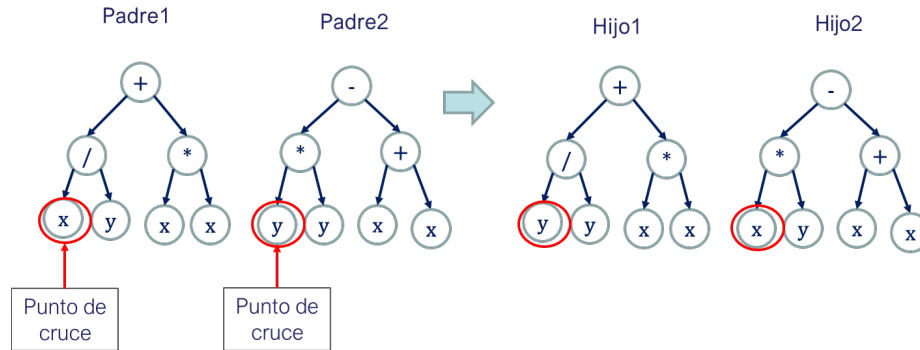


Figura 39. Cruza con poco intercambio genético

Los puntos de cruce no deberían seleccionarse con probabilidad uniforme. Dado que la mayoría de los nodos serán hojas, la selección uniforme seleccionara más frecuentemente nodos hojas llevando a un intercambio de poco material genético. Para contrarrestar esto se sugiere que el 90% de las veces se seleccionen funciones y el 10% restante terminales.

Mutación

Es un operador genético asexual por lo que solo requiere la selección de 1 padre y como resultado se tiene un solo hijo.

El libro clásico de Koza sobre GP de 1992 [4] aconseja a los usuarios establecer la probabilidad de mutación (P_m) en 0, es decir, sin mutación. Más recientemente, Banzhaf *et al.* recomiendan 5% probabilidad de mutación [6]. La cruce en algunos casos funciona como mutación de un punto.

Mutación de subárbol

- I. Elegir un punto de mutación al azar, con probabilidad uniforme, dentro del individuo seleccionado.
- II. Eliminar el subárbol cuya raíz es el punto de mutación.
- III. Inserta un nuevo subárbol generado aleatoriamente en ese punto.

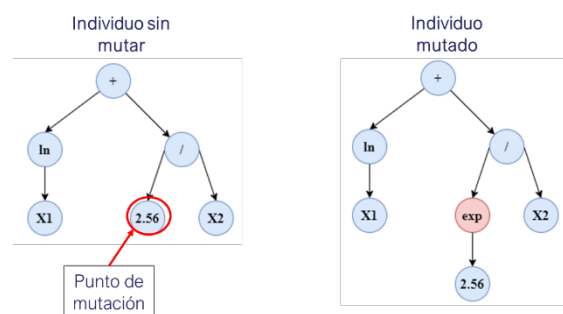


Figura 40. Mutación de subárbol

Mutación de 1 punto

Consiste en cambiar el valor de un nodo del árbol seleccionado al azar.

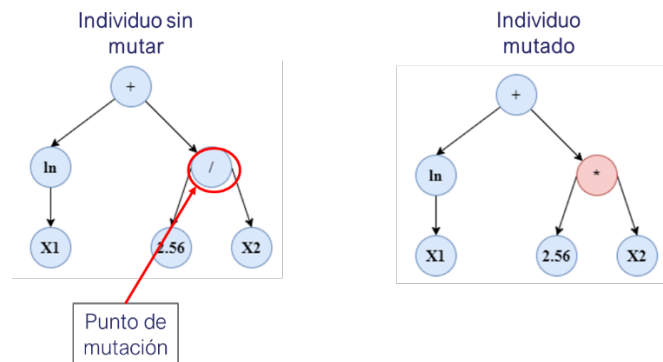


Figura 41. Mutación de 1 punto

Mutación de intercambio

Consiste en seleccionar un nodo intermedio del árbol al azar e intercambiar de posición sus ramas.

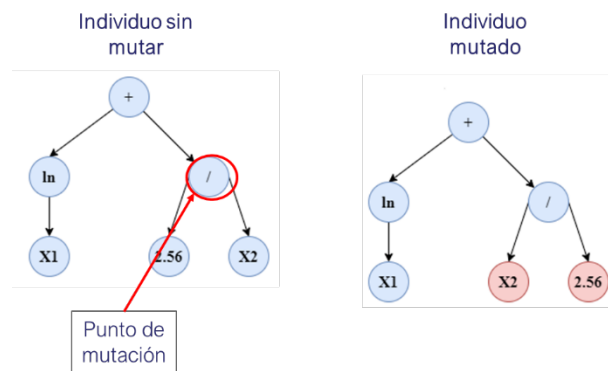


Figura 42. Mutación por intercambio

La elección de cuál operador genético usar para crear una descendencia es probabilística puede variar. Los operadores en PG son mutuamente excluyentes (a diferencia de otros algoritmos evolutivos donde la descendencia a veces se obtiene a través de una composición de operadores). A continuación, se muestran 2 combinaciones que suelen usarse:

Configuración 1

$$P_c = 90\%$$
$$P_r = 10\%$$

Configuración 2

$$P_c = 90\%$$
$$P_m = 5\%$$
$$P_r = 5\%$$

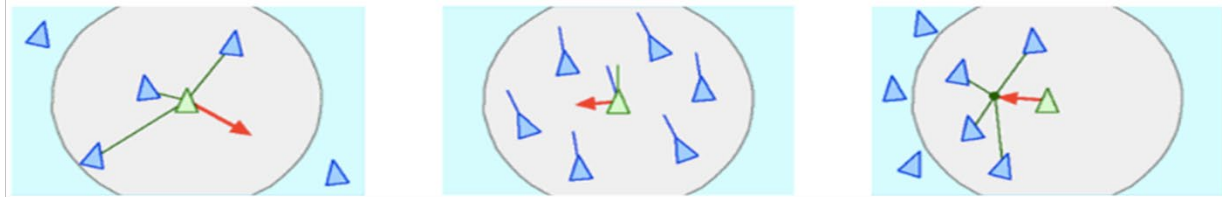
$$P_c + P_m + P_r = 1$$

Figura 43. Combinación de operador genéticos en PG

5. Enjambre de partículas

Antecedentes: Boids

En 1987, Craig Reynolds creó un modelo de movimiento animal coordinado en el que los agentes (boids) obedecían tres reglas locales simples, ahora conocidas como **reglas de Reynolds** [7].



1. **Separación:** maniobrar para evitar acercarse demasiado a los demás.

2. **Alineación:** maniobrar hacia la dirección promedio de los demás

3. **Cohesión:** maniobrar hacia la posición promedio de los demás.

Figura 44. Reglas de Reynolds

Esta receta logra un comportamiento de enjambre realista, con algunos ajustes de parámetros necesarios dependiendo de la especie simulada.

Es importante notar que cada *boid* tiene su propio campo de percepción, es decir, que solo puede ver a cierta distancia y con un campo de visión específico (i.e., los *boids* no pueden ver detrás de ellos), por tanto, los ajustes que hace a su velocidad en cualquier momento son en función de las posiciones y velocidades de los cuerpos en su campo de percepción, más que una función de toda la bandada.

Es importante tener en cuenta que estas reglas no están estrictamente inspiradas en la naturaleza, ya que Reynolds no intentaba explicar el comportamiento natural del enjambre, simplemente intentaba emularlo. Varios autores señalaron que "muchas personas que ven las simulaciones, las reconocen de inmediato como una representación de una bandada natural y las encuentran igualmente agradables de ver" (Ver video en <https://www.youtube.com/watch?v=QbUPfMXXQIY>).

Boids + comportamiento de anidado

En 1995, Kennedy y Eberhart [8] incluyeron un dormitorio (*roost*) o, más generalmente, un punto de atracción (ejemplo, una presa) en una simulación similar a *Boids* simplificada, tal que cada agente:

- Se siente atraído por la ubicación del dormitorio,
- Recuerda cuando estuvo más cerca del dormitorio,
- Comparte información con sus vecinos sobre su ubicación más cercana al dormitorio.

Optimización de Enjambre de Partículas (PSO)

PSO consiste en un sistema multiagente (partículas) que simulan el comportamiento de enjambre de las aves.

- Una **partícula** se iguala con una solución candidata a un problema de optimización. $x_i(t)$ denota la partícula i en el tiempo t . Cada partícula denotada por $x_i(t)$ representa una solución dentro del espacio de búsqueda.
- Tal partícula tiene una posición y una velocidad.
- Su velocidad es un vector de desplazamiento en el espacio de búsqueda, que (se espera) contribuirá a un cambio fructífero en su posición en la siguiente iteración.
- El valor de aptitud de cada partícula representa la calidad de su posición en el panorama de optimización.

Ejemplo:

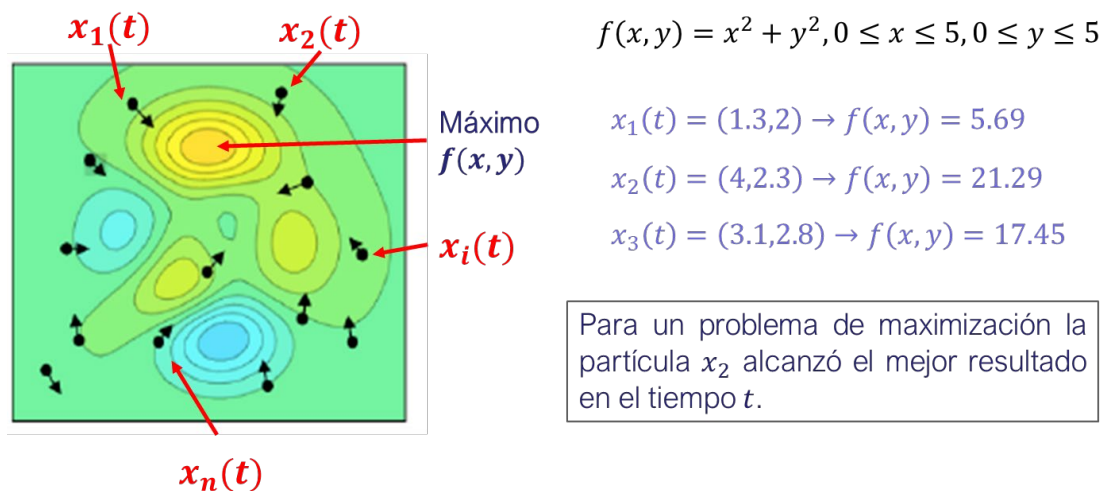


Figura 45. Ejemplo de partículas en PSO

Las partículas se mueven sobre el espacio de búsqueda con una cierta velocidad. Cada partícula tiene: estado interno + estado del vecindario (red de conexiones sociales).

En cada paso de tiempo, la velocidad (tanto la dirección como la rapidez) de cada partícula se ve influenciada por:

pbest: su propia mejor posición encontrada hasta ahora

lbest: la mejor solución encontrada hasta ahora por los compañeros de equipo en su vecino social,

gbest: la mejor solución global hasta el momento

más un elemento de aleatoriedad.

Ejemplo:

$$f(x, y) = x^2 + y^2, 0 \leq x \leq 5, 0 \leq y \leq 5$$

$x_1(t) = (1.3, 2) \rightarrow f(x, y) = 5.69$	Vecindario
$x_2(t) = (4, 2.3) \rightarrow f(x, y) = 21.29$	$\mathcal{N}(x_1) = (x_3, x_5, x_6)$
$x_3(t) = (1, 5) \rightarrow f(x, y) = 26$	$pbest(x_1) = 5.69$
$x_4(t) = (3.1, 2.8) \rightarrow f(x, y) = 17.45$	$lbest(x_1) = 28.8$
$x_5(t) = (4.8, 2.4) \rightarrow f(x, y) = 28.8$	$gbest(x_1) = 29.84$
$x_6(t) = (2.5, 2.5) \rightarrow f(x, y) = 12.5$	
$x_7(t) = (5, 2.2) \rightarrow f(x, y) = 29.84$	
$x_8(t) = (1.9, 3.4) \rightarrow f(x, y) = 15.17$	

Figura 46. Ejemplo de partículas con $pbest$, $gbest$ y $lbest$

Pasos de PSO:

1. Definir función objetivo.
2. Definir parámetros de ejecución.
 - Número de partículas N
 - Número de pasos de tiempo
 - $a \rightarrow$ inercia
 - $b_1 \rightarrow$ factor de aprendizaje (influencia propia)
 - $b_2 \rightarrow$ factor de aprendizaje (influencia social)
 - Número de vecinos (versión local)
3. Inicialización aleatoria de posiciones y velocidades.
4. Evaluar la función de aptitud con las posiciones actuales de las partículas.
5. Asignar los mejores resultados ($pbes$, $lbest/gbest$).
6. Repetir hasta alcanzar condición de paro:
 - I. Actualizar la posición de cada partícula de acuerdo con 1 de las siguientes formulas

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

$$v_i(t+1) = a * v_i(t) + c1 * r1(pbest - x_i(t)) + b2 * r2(gbest - x_i(t)) \text{ (versión global)}$$

$$v_i(t+1) = a * v_i(t) + c1 * r1(pbest - x_i(t)) + b2 * r2(lbest - x_i(t)) \text{ (versión local)}$$

II. Evaluar la función de aptitud con las nuevas posiciones de las partículas.

III. Actualizar los mejores valores (pbest, lbest/gbest).

r_1 y r_2 son valores aleatorios que toman valores entre 0 y 1

El parámetro α llamado inercia, controla la fracción en el desplazamiento con la misma dirección que la partícula llevaba en el paso previo. Un valor de inercia alto presiona hacia la exploración global en una nueva área de búsqueda, mientras que un valor pequeño presiona hacia el ajuste en el área de búsqueda actual. Se sugieren valores entre [0.4,0.9] [9].

El parámetro b_1 controla la fracción del desplazamiento en dirección de la mejor solución encontrada por la partícula.

El parámetro b_2 controla la fracción del desplazamiento en dirección de la mejor solución encontrada por el vecindario (versión local) o por el enjambre (versión global). Se sugiere que la suma $b_1+b_2=4$ [9].

6. Colonia de hormigas

Optimización por Colonia de Hormigas (ACO de sus siglas en inglés) es una técnica de búsqueda general basada en poblaciones para la solución de problemas combinatorios difíciles, que se inspira en el comportamiento de rastros de feromonas de colonias de hormigas reales.

En ACO, un conjunto de agentes de software llamados hormigas artificiales buscan buenas soluciones a un problema de optimización dado. Para aplicar ACO, el problema de optimización se transforma en el problema de encontrar el mejor camino en un grafo ponderado.

La primera versión de ACO fue propuesta por Dorigo [10]. Inicialmente, las hormigas se distribuyen aleatoriamente en los nodos de un grafo. Cada hormiga artificial elige una arista que sale de su ubicación de acuerdo con una regla que toma en cuenta la longitud y la cantidad de feromonas de cada arista.

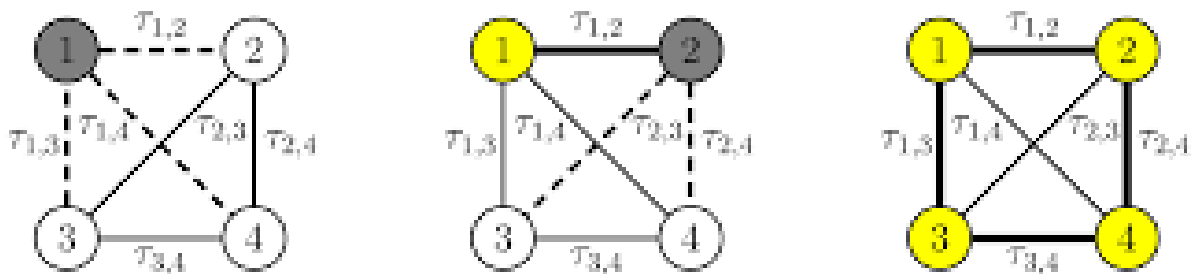


Figura 47. Ejemplo de recorrido de ACO

- Las aristas que llevan a nodos ya visitados por la hormiga no son consideradas en la selección del siguiente camino.

- Una vez que todas las hormigas realizan un recorrido completo por el grafo, cada una de ellas vuelve sobre su propio recorrido depositando en las aristas recorridas una cantidad de feromonas inversamente proporcional a la longitud del recorrido.
- Antes de iniciar una nueva búsqueda, las feromonas de todas las aristas se evaporan en una pequeña cantidad.

Cada hormiga artificial es un agente con las siguientes características:

1. Prefiere caminar por las aristas con valor de feromonas,
2. Los nodos ya visitados están prohibidos hasta que se complete el circuito (lista tabú)
3. Cuando finaliza el viaje, los valores de feromonas se actualiza en cada arista usada.

El flujo operativo básico en un sistema de hormigas es el siguiente:

Paso 1: representar el espacio de soluciones mediante un grafo con pesos.

Paso 2: inicializar los parámetros.

Paso 3: generar soluciones aleatorias a partir de la caminata aleatoria de cada hormiga.

Paso 4: actualiza las intensidades de las feromonas.

Paso 5: ir al paso 3 y repetir hasta que se satisfaga una condición de parada

Para la selección del siguiente nodo en el recorrido de una hormiga se usa un método basado en probabilidades. Supongamos que la hormiga k en el paso de tiempo t realiza una caminata aleatoria desde el nodo i hacia el siguiente nodo j . Esta decisión se toma de acuerdo con la probabilidad de transición definida por:

$$P_k(i, j) = \begin{cases} \frac{\tau(i, j)^a \cdot \eta(i, j)^b}{\sum_{c_{il} \in N_k} \tau(i, l)^a \cdot \eta(i, l)^b} & \text{si } c_{ij} \in N_k \\ 0 & \text{de otra forma} \end{cases}$$

Donde:

- $\tau(i, j)^a$ es la intensidad de la feromona en la arista (i, j) .
- $\eta(i, j)^b$ es la visibilidad de la arista (i, j) . Generalmente, el inverso de la distancia de la arista (i, j) . $\eta(i, j) = 1/d(i, j)$.
- l es un nodo que no ha sido visitado por la hormiga k .
- N_k es el conjunto de nodos permitidos para la hormiga k que se encuentra en el nodo i (vecindario).

- a, b son los parámetros de control que determinan la importancia relativa de la feromona frente a la visibilidad.
- Una vez calculadas las probabilidades usarlas para aplicar un método de selección como la ruleta

Cuando todas las hormigas de la colonia han logrado su solución, el rastro de feromonas se actualiza para todas las aristas mediante la regla de actualización global de feromonas definida por:

$$\tau_{t+1}(i, j) = (1 - \rho)\tau_t(i, j) + \sum_{k=1}^m \Delta t_k(i, j)$$

Donde:

- $1 - \rho$ representa la evaporación de las feromonas.
- $\Delta t_k(i, j)$ es la cantidad de feromona depositada por hormiga k .

$$\Delta t_k(i, j) = \begin{cases} \frac{Q}{L_k} & \text{si la hormiga } k \text{ viaja por la arista } (i, j) \\ 0 & \text{de otra forma} \end{cases}$$

- Q es una constante, usualmente 1.
- L_k es la distancia total recorrida por hormiga k .

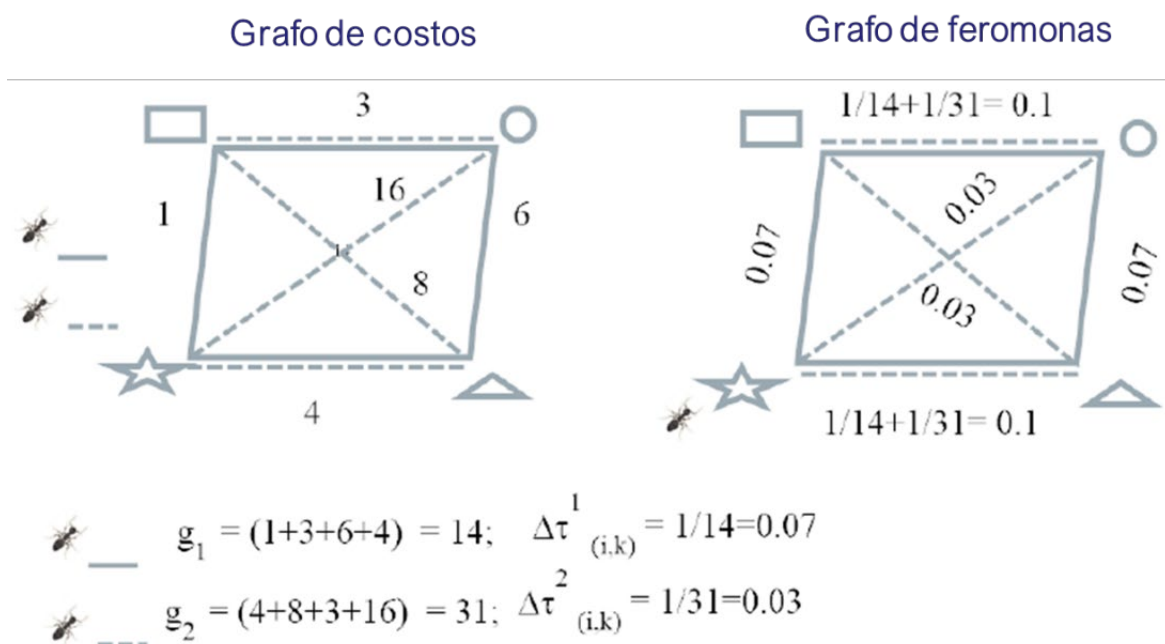


Figura 48. Ejemplo de ACO

7. Colonia de abejas

Colonia de abejas artificiales (ABC de sus siglas en inglés) es un algoritmo basado en enjambres que fue introducido originalmente por Karaboga [1], inspirado en el comportamiento inteligente de las abejas en el forrajeo. Este se basa en 3 componentes principales:



Figura 49. Componentes de ABC

ABC se basa en el comportamiento de forrajeo de las abejas. El enjambre está conformado por:

- Abejas obreras
- Abejas observadoras (Onlooker)
- Abejas exploradoras (Scout)

El rol de cada abeja puede cambiar a lo largo de la ejecución del programa. La Figura 50 muestra el diagrama de flujo del algoritmo ABC.

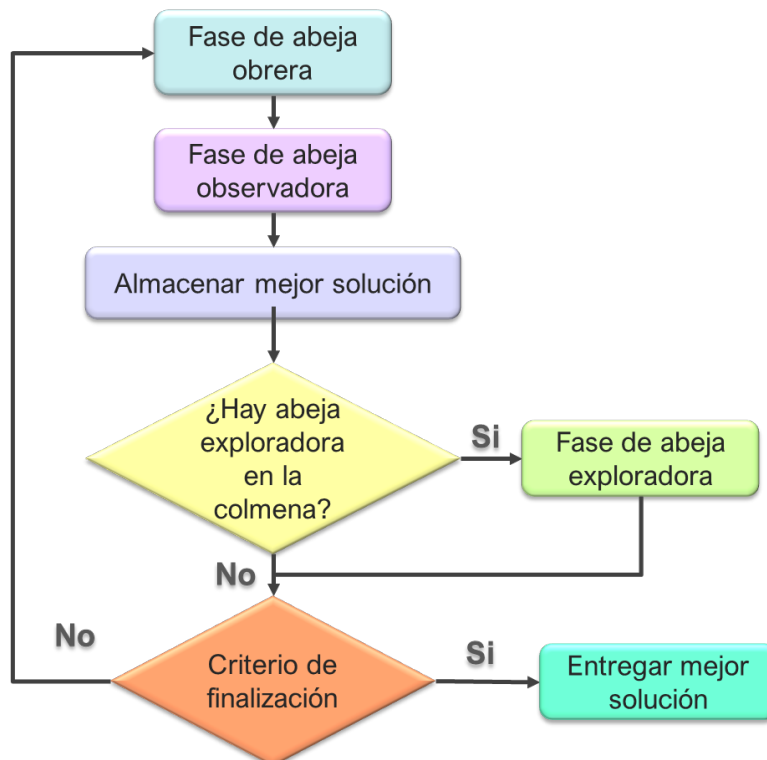


Figura 50. Esquema de algoritmo ABC

Fase de inicialización

- Cada fuente de comida representa una solución al problema a optimizar.
- La cantidad de néctar de una fuente de comida corresponde a la calidad (*fitness*) de la solución asociada a esa fuente.
- El número de abejas obreras es igual al número de soluciones.

Cada solución x_i ($i = 1, 2, \dots, D$) es un vector D dimensional donde cada posición es una de las variables/parámetros a optimizar.

Se debe generar una población inicial aleatoria de soluciones de tamaño **SN** (fuente de comida). **SN** denota el tamaño de la población de soluciones usando la siguiente ecuación:

$$x_{ij} = l_j + r_1 * (u_j - l_j)$$

Donde:

$i \rightarrow$ índice del número de soluciones

$j \rightarrow$ índice del número de variables a optimizar

$r_1 \rightarrow$ valor aleatorio $[0,1]$ (distribución uniforme)

$u_j \rightarrow$ límite superior de la j -ésima variable

$l_j \rightarrow$ límite inferior de la j -ésima variable

Parámetros de control

1. **Abejas obreras:** 50% del enjambre. Es asignada a una fuente de comida y provee información del vecindario en la fuente almacenada en su memoria.
2. **Abejas observadoras:** 50% del enjambre. Obtienen información de las abejas empleadas y seleccionan una fuente de comida para recolectar néctar.
3. **Abejas exploradoras:** las abejas cuya fuente de comida se agotó se convierten en exploradoras.
4. **Tamaño del enjambre.**
5. **Límite:** indica la cantidad de veces permitida en que no se logra mejorar una solución. Generalmente se define como $\frac{SN}{2} \times D$ donde SN es el tamaño del enjambre.
6. **Número de ciclos.**

Se toman las siguientes consideraciones:

- El número de abejas obreras es igual al número de fuentes de comida.
- Cada fuente de comida es una posible solución al problema

- La cantidad de néctar de una fuente de comida es igual a la calidad de la solución (*fitness*)

Fase de abeja obrera

Una abeja artificial obrera produce probabilísticamente una modificación en la posición (solución) en su memoria para encontrar una nueva fuente de alimento y prueba la cantidad de néctar (valor de aptitud) de la nueva fuente (nueva solución).

Las abejas obreras seleccionan aleatoriamente una posición de fuente de alimento y producen una modificación en la existente en su memoria como se describe en:

$$v_{ij} = x_{ij} + r_2 * (x_{ij} - x_{kj})$$

Donde $k \in \{1, 2, \dots, SN\}$ y $j \in \{1, 2, \dots, D\}$ son índices seleccionados aleatoriamente y $k \neq i$. Si la nueva solución excede el valor permitido de alguna variable, se asigna el valor mayor/menor permitido.

$v_{ij} \rightarrow$ la j -ésima posición de la solución i actualizada

$x_{ij} \rightarrow$ la j -ésima posición de la solución i

$x_{kj} \rightarrow$ la j -ésima posición de otra solución k seleccionada al azar

$r_2 \rightarrow$ número aleatorio entre $[-1, 1]$ (distribución uniforme)

- Si el parámetro de **límite** se alcanza, la fuente de comida se abandona.
- Si la cantidad de néctar (*fitness*) de la nueva solución es mayor que la anterior, la abeja memoriza la nueva solución y olvida la anterior. Inicializar el parámetro **límite** a cero.
- Después que todas las abejas obreras completan el proceso de búsqueda, comparten la información de las fuentes de comida con las abejas observadora.

Fase de abeja observadora

En esta fase una abeja observadora evalúa la información de néctar tomada de todas las abejas obreras y elige una fuente de alimento con una probabilidad relacionada con su cantidad de néctar, calculada con la siguiente expresión:

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n}$$

Para casos de maximización fit_i es la evaluación directa de la solución en la función objetivo.

Para casos de minimización se usa

$$fit_i = \begin{cases} \frac{1}{1 + f(i)} & \text{si } f(i) \geq 0 \\ 1 + |f(i)| & \text{si } f(i) < 0 \end{cases}$$

Donde $f(i)$ es la evaluación directa de la solución en la función objetivo. Una vez que se tiene calculadas las probabilidades de selección de cada fuente de comida (abeja obrera), se realiza la selección usando cualquier método de selección basado en la aptitud como la ruleta para que cada

observadora decida a que obrera debe seguir. Al usar un método basado en el valor de fitness de una solución, las observadoras tenderán a seguir a aquellas obreras con mejores soluciones.

Luego que la abeja observadora elige una fuente de comida, se crea una fuente de comida cercana usando la misma expresión que la abeja obrera, es decir que la observadora colabora en la explotación de la solución asociada a la abeja obrera que decidió seguir. Se evalúa la nueva fuente de comida y se modifica la fuente de comida si la solución encontrada por la observadora es mejor que la anterior. Para modificar la fuente de comida se usa la misma expresión que usan las abejas obreras:

$$v_{ij} = x_{ij} + r_2 * (x_{ij} - x_{kj})$$

Exploración contra explotación

Las abejas obreras y observadoras realizan el proceso de explotar una fuente de comida, mientras que las abejas exploradoras llevan el proceso de explorar nuevas fuentes de comida.

Fase de abeja exploradora

Las abejas libres que eligen sus fuentes de alimento al azar se llaman exploradoras, esto quiere decir que las abejas obreras cuyas soluciones no mejoran luego de un número predeterminado (parámetro límite), se convierten en exploradoras, sus soluciones son abandonadas y se generan nuevas soluciones aleatorias usando la misma expresión que se usa para generar la población inicial:

$$x_{ij} = l_j + r_1 * (u_j - l_j)$$

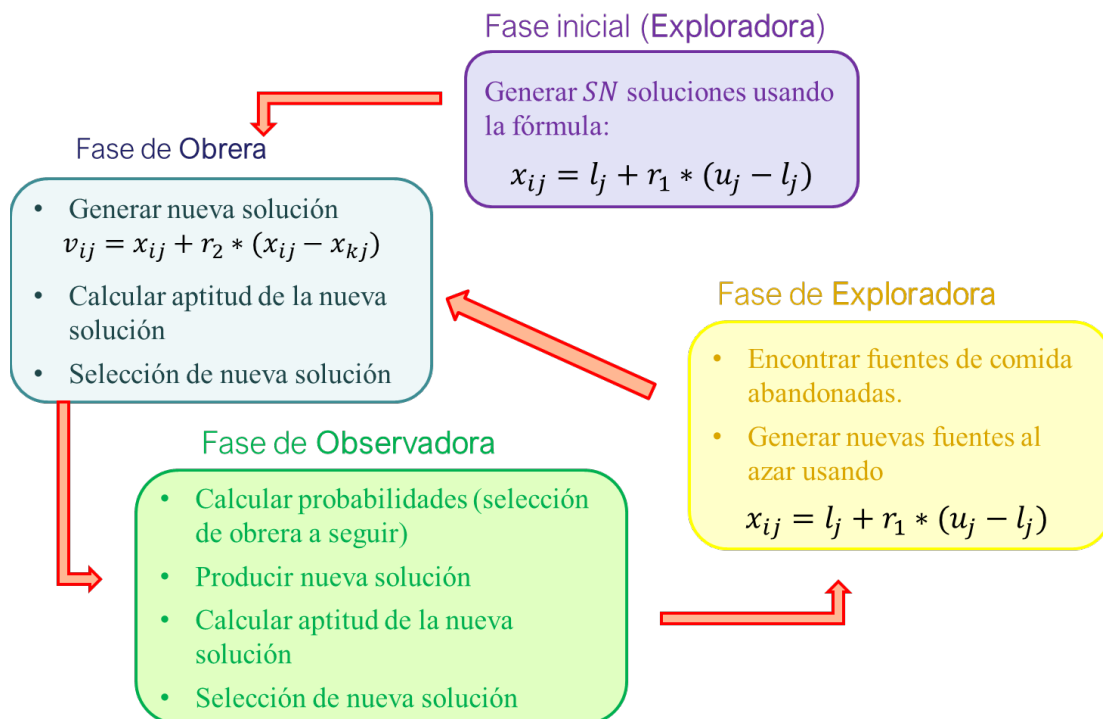


Figura 51. Diagrama de ejecución de ciclos en ABC

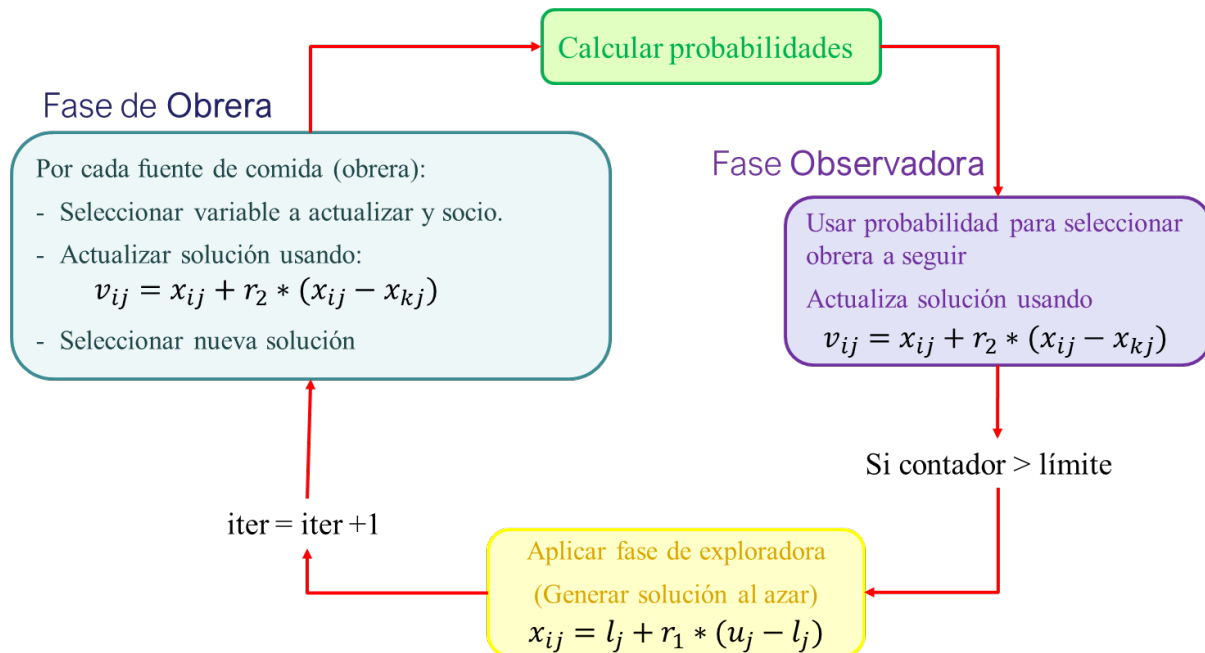


Figura 52. Diagrama de fase de obrero

Ejercicios

1. Ejercicio Optimización y Espacios de Búsqueda

Actualmente hay mucha investigación en la producción de vehículos autónomos que se pueden utilizar en carreteras reales. Para cada una de las siguientes capacidades que debería exhibir dicho sistema, indique si se trata de un problema de optimización, modelado o simulación:

- Aprender a reconocer las señales de tráfico.
- Reconocer una señal de tráfico en una señal de video mientras el vehículo avanza.
- Planificación de la ruta más corta o rápida entre dos lugares.
- Evitar a un niño que corre en la carretera.

2. Ejercicios de Algoritmos Genéticos

a) Dados los siguientes individuos: 101011100110100 y 110110010101101. ¿Cuáles son los 2 hijos válidos si se aplica el operador de cruce de 2 puntos con puntos de corte 4 y 11?

☐ 111010101110101

☐ 110110101111001

☐ 110111100111101

☐ 101010010100100

☐ 101000101011001

b) Usando los datos de la tabla, calcular la probabilidad de selección para cada individuo con el método de la ruleta.

	Aptitud ($f(x)$)
Individuo 1	5
Individuo 2	12
Individuo 3	6
Individuo 4	3
Individuo 5	10
Individuo 6	7
Individuo 7	4
Individuo 8	9

c) Dado el siguiente individuo 1010011001 y el vector de probabilidades (0.03, 0.5, 0.75, 0.02, 0.2, 0.7, 0.3, 0.4, 0.09, 0.2). Realizar la mutación del individuo por inversión de bit con una probabilidad de mutación de 0.1.

☐ 1010001001

☐ 0011011011

☐ 1010011001

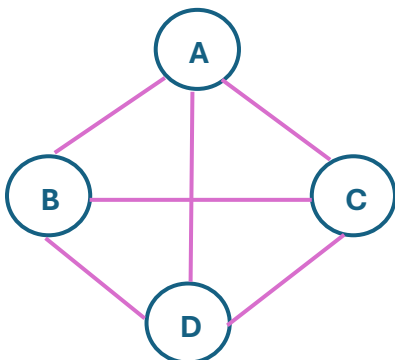
☐ 0011000111

☐ 0010011001

3. Ejercicio de programación genética

Dado el conjunto de funciones $F=\{+, -, *, /, \text{sen}, \text{cos}, \text{tan}, \ln\}$ y el conjunto de terminales $T=\{x, y, z, \text{randint}, \text{rand}, 3.14, 1\}$, el límite de profundidad $d_{max} = 3$ y un tamaño de población $M = 12$. Crear una población inicial válida de individuos usando el método Ramped Half and a Half. Dibujar los árboles correspondientes a cada individuo.

4. Ejercicio Hormigas



Caminos	Costo	Feromonas (τ)
A-B	5	0.1
A-C	3	0.1
A-D	7	0.1
B-C	9	0.1
B-D	6	0.1
C-D	4	0.1

Realizar una corrida del algoritmo de colonia de hormigas asumiendo 1 hormiga en cada nodo. Considerando el grafo y la tabla dados:

a) Determinar el recorrido de la hormiga que sale de nodo C.

b) Considerando que los siguientes recorridos de las otras hormigas: $H_A = \{A, D, C, B, A\}$, $H_B = \{B, C, A, D, B\}$ y $H_D = \{D, A, C, B, D\}$. Actualizar los valores de feromonas.

Usar la ruleta para la selección de a que nodo se mueve la hormiga usando probabilidad de selección calculada con la siguiente formula:

$$P_k(i, j) = \begin{cases} \frac{\tau(i, j)^a \cdot \eta(i, j)^b}{\sum_{c_{il} \in N_k} \tau(i, l)^a \cdot \eta(i, l)^b} & \text{si } c_{ij} \in N_k \\ 0 & \text{de otra forma} \end{cases}$$

Usar $\rho = 0.1$, $Q = 1$, $a = 0.3$ y $b = 0.7$

Actualizar las feromonas de cada arista usando las siguientes formulas:

$$\tau_{t+1}(i, j) = (1 - \rho)\tau_t(i, j) + \sum_{k=1}^m \Delta t_k(i, j) \text{ usar } 1 - \rho = 0.1$$

$$\Delta t_k(i, j) = \begin{cases} \frac{Q}{L_k} & \text{si la hormiga } k \text{ viaja por la arista } (i, j) \\ 0 & \text{de otra forma} \end{cases}$$

5. Ejercicio Abejas

Minimizar la ecuación $f(x, y) = x^2 - \frac{1}{y}$ donde $-5 \leq x, y \leq 5$ usando colonia de abejas con los siguientes parámetros (realizar 1 iteración del algoritmo):

Tamaño del enjambre = 10 límite = 2

$$v_{ij} = x_{ij} + r_2 * (x_{ij} - x_{kj}) \quad \text{fit}_i = \begin{cases} \frac{1}{1 + f(i)} & \text{si } f(i) \geq 0 \\ 1 + |f(i)| & \text{si } f(i) < 0 \end{cases}$$

El ejercicio esta completo cuando cada observadora siga a una obrera, explote su solución y de ser necesario se ejecute el ciclo de exploradora. Debe quedar clara la simulación de la ruleta para escoger a la obrera que decide seguir la observadora y como se actualizan las soluciones tanto en los ciclos de obreras como de observadoras.

6. Ejercicio de Optimización con PSO.

Dadas las trayectorias que siguieron 3 partículas (en un espacio bidimensional), calcule la nueva velocidad, la posición final y el valor de la función objetivo. Indique claramente que valores tienen los vectores Pbest y Gbest.

Id partícula	Trayectoria		Función objetivo	Velocidad final	
	x_1	x_2		v_{x_1}	v_{x_2}
1	1.5	2.2	7.09	1.5	0.3
	-4.5	0.2	20.29		
	1.6	-2.3	7.85		
2	3.4	-1.1	12.77	2.5	3.2
	1.8	0.9	4.05		
	-2.5	0.2	6.29		
3	7.3	-6.5	95.54	6.5	7.2
	0.4	2.8	8		
	-3.2	-4.6	31.4		

Considere los siguientes valores de parámetros

$$a = 0.6, b1 = 0.8, b2 = 1.2$$

Use la ecuación $v_i(t + 1) = a * v_i(t) + c1 * r1(pbest - x_i(t)) + b2 * r2(gbest - x_i(t))$

Id partícula	Nueva velocidad		Nueva posición		Valor función
	v_{x_1}	v_{x_2}	x_1	x_2	
1					$f(x_1, x_2) = x_1^2 + x_2^2$
2					
3					

Referencias

1. Riquelme Medina, I. (2014). Algoritmos bioinspirados: Una revisión según su fundamento biológico. Tesis.
2. Eiben, A. E. & Smith, J. E. (2015). Introduction to Evolutionary Computing (2nd ed.). Springer. ISBN: 978-3-662-44874-8.
3. S. N. Sivanandam and S. N. Deepa. (2010). Introduction to Genetic Algorithms (1st. ed.). Springer Publishing Company, Incorporated.
4. Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., & Lanza, G. (2006). Genetic programming IV: Routine human-competitive machine intelligence, vol. 5, Springer Science & Business Media.
5. Poli, R., Langdon, W.B. & McPhee, N.F. (2008). A Field Guide to Genetic Programming. Lulu Enterprises, UK Ltd.
6. Banzhaf, W., Nordin, P., Keller, R.E., & Francone, F.D. (1998). Genetic Programming: An Introduction. Morgan Kaufmann, San Francisco.
7. Reynolds, C. (1987). Flocks, herds and schools: A distributed behavioral model. Computer Graphics, vol. 21(4), pp. 25–34.
8. J. Kennedy, R.C. Eberhart. (1995). Particle swarm optimization. In Proceedings of IEEE International Conference on Neural Networks, vol. 4, pp.1942-1948.
9. Keller, J.M., Liu, D., & Fogel, D.B. (2016). Fundamentals of computational intelligence: neural networks, fuzzy systems, and evolutionary computation. John Wiley & Sons.
10. Dorigo, M. (1992). Optimization, learning and natural algorithms. Doctoral dissertation, Politecnico di Milano, Milan, Italy.
11. Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization (Vol. 200, pp. 1-10). Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.