CAPÍTULO 2: COMPUTACIÓN EVOLUTIVA

2. INTRODUCCIÓN

El principio de la evolución [58] se origina en el concepto básico de la biología, donde cada criatura en la cadena es el producto de una serie de "accidentes" que se han elegido bajo la presión selectiva que ejerce el medio ambiente. Una gran cantidad de generaciones, la variación aleatoria y la selección natural muestran el comportamiento de individuos y especies para ajustarse a las demandas de sus vecinos.

Este ajuste puede ser bastante extraordinario y determinante, una clara indicación de esto es la creatividad de la evolución. Mientras que la evolución no tiene un propósito intrínseco (ésta es, meramente el efecto de la actuación de las leyes físicas sobre y con las poblaciones y especies), sino que es capaz de dar soluciones ingenieriles a los problemas de sobrevivencia que son únicas para las circunstancias de cada individuo y bastante ingeniosas.

El proceso evolutivo dentro de una computadora provee un medio para solucionar problemas ingenieriles complejos (involucrando disturbios caóticos, aleatoriedad y dinámicas no lineales complejas) que los algoritmos tradicionales no han sido capaces de conquistar [7, 8, 108, 128]. Por esta razón, el campo de la computación evolutiva es una de las áreas de más rápido crecimiento de la ciencia de la computación y de la ingeniería. Es decir, apunta a muchos problemas que fueran previamente alcanzados, tales como un diseño rápido de medicinas, y el análisis veloz de tácticas de combate para defensa. Potencialmente, este campo puede completar el sueño de la inteligencia artificial: una computadora que pueda aprender y se convierta en un experto en cualquier área elegida.

En términos más generales, la evolución puede describirse como un proceso iterativo de dos pasos, consistentes de una variación aleatoria seguida de una selección. El enlace entre esta descripción de la evolución y los algoritmos de optimización que son la estrella de la computación evolutiva es conceptualmente simple.

Al igual que la evolución natural que comienza desde una población inicial de criaturas, la opción algorítmica comienza al seleccionar un conjunto inicial de

soluciones para un problema en particular. El conjunto puede ser elegido al generar soluciones aleatorias o al utilizar cualquier conocimiento disponible sobre el problema.

Estas soluciones denominadas "*padres*" entonces generan "*hijos*" por medio de variaciones aleatorias. Estas soluciones resultantes se evalúan en función de su efectividad o aptitud (su *fitness*) y la selección a las que son sometidas. Tal como en la naturaleza se impone la regla "sobrevive el más fuerte", entonces las soluciones de menor fitness se eliminan, y el proceso es repetido sobre las sucesivas generaciones.

Los algoritmos tradicionales usados para descubrir las soluciones más apropiadas (algoritmos de optimización) requieren, para ser útiles, que sus usuarios realicen muchas asumpciones de cómo evaluar el fitness de una solución. A estos medios de evaluación tradicionales se los puede denominar: fitness, función de costo, superficie de respuesta, o, en ingeniería índice de performance. Por ejemplo, los algoritmos de programación lineal demandan que las funciones de costo también sean lineales. Otra opción tradicional, es la búsqueda basada en el gradiente; en el cual el punto de gradiente cero indica que el máximo o el mínimo es alcanzado. Esta última opción requiere una función de costo diferenciable.

Pero los algoritmos evolutivos no requieren tales asumpciones. Fundamentalmente, el índice de *performance* necesita ser capaz de ordenar de acuerdo a un rango a dos soluciones competidoras, esto significa que debe determinar cuál solución es la mejor de ambas. Esto hace que una amplia variedad de problemas que están fuera del rango de la ingeniería convencional sean atacados por la opción evolutiva. Concluyendo, los algoritmos evolutivos pueden a menudo resolver problemas que las técnicas numéricas no logran solucionar.

La alternativa evolutiva ofrece grandes ventajas. Una de ellas es la capacidad para adaptarse a situaciones cambiantes. Por ejemplo, el caso en que un administrador deba encontrar la mejor planificación para la operación de una fábrica. Sin dudas, habrá muchas restricciones: disponibilidad de personal, el número de máquinas, el tiempo requerido para configurar los cambios en las máquinas, entre otras. Aún si el administrador encontrase la planificación óptima,

sería deseable contar con planificaciones alternativas que tuviesen en cuenta, por ejemplo, la rotura de máquinas, la disminución del personal, etc.; para poder obtener el o los productos a tiempo. En el mundo real, el problema puede ser significativamente divergente del planteado originalmente.

Desafortunadamente, en muchos de los procedimientos de optimización tradicionales, el cálculo debe ser iniciado nuevamente desde el comienzo si una de las variantes del problema cambia. Esto es computacionalmente muy caro. Con un algoritmo evolutivo la población actual actúa como una reserva de conocimiento almacenado que puede ser aplicado a ambientes dinámicos. No siendo necesario iniciar el cálculo desde ningún punto.

Otra ventaja que presenta la opción evolutiva es la siguiente: puede generar soluciones lo suficientemente buenas rápidamente para su uso. Esta habilidad es bien ilustrada en el problema del viajante. Este problema pertenece a la familia de los problemas NP-duros, donde NP hace referencia a los problemas donde no puede hallarse un algoritmo que lo resuelva en tiempo polinomial. Por lo que puede ser imposible encontrar el óptimo global en algunas aplicaciones, o no haber una manera de chequear si una solución es un óptimo global. En muchos casos encontrar el óptimo global puede ser innecesario. Pero hallar una buena solución (un óptimo local) velozmente puede ser más deseable que encontrar la mejor solución (el óptimo global) lentamente.

Desde hace aproximadamente treinta años [107] existe un creciente interés en los sistemas de resolución de problemas basados en los principios de evolución y hereditarios, tales como: mantener una población de soluciones potenciales (que como se menciona anteriormente sufren algún proceso de selección basado en el fitness de individuos), y algunos operadores “genéticos”. Una clase de tales sistemas son las *estrategias evolutivas*. Estos algoritmos imitan los principios de la evolución natural para problemas de optimización de parámetros [114, 126] (Rechenberg, Schwefel). Otro tipo es la *programación evolutiva* [51] que introduce Fogel; es una técnica para buscar a través de un espacio pequeño de máquinas de estados finitos. Las técnicas de búsqueda *Scatter* que han sido presentadas por Glover [64] mantienen una población de puntos de referencia y genera vástagos por combinaciones lineales de peso. Otro tipo de sistemas

basados en la evolución son los *algoritmos genéticos* introducidos por Holland 1975 [82]. En 1990, Koza [95] propone la *programación genética*, mediante la cual se busca el mejor programa computacional para resolver un problema en particular.

En este capítulo se describen las estrategias evolutivas, la programación evolutiva y la programación genética en las secciones 2.3, 2.4 y 2.5 respectivamente. Mientras que, los algoritmos genéticos son tratados con mayor profundidad en el capítulo 3.

3. DESCRIPCIÓN DE UN ALGORITMO EVOLUTIVO GENÉRICO

Se usa el término Algoritmo Evolutivo, para todos los sistemas basados en la evolución (incluyendo los sistemas descriptos en la sección anterior).

El algoritmo evolutivo es probabilístico y mantiene una población de individuos, $P(t) = \{x^t_1, \ldots\ldots\ldots\ldots, x^t_n\}$ para la iteración t [107]. Cada individuo (o cromosoma) representa una solución potencial al problema en cuestión, y, un algoritmo evolutivo es implementado como alguna estructura de datos S. Cada solución x^t_i es evaluada para dar alguna medida de su "fitness". Entonces, una nueva población (iteración t+1) es formada al seleccionar los individuos de mejor fitness (seleccionar). Algunos miembros de la nueva población sufren transformaciones por medio de operadores "genéticos" (alterar) para construir nuevas soluciones. Existen transformaciones unarias m_i (mutación), las cuales crean nuevos individuos al realizar pequeños cambios en un único individuo ($m_i : S \rightarrow S$), y transformaciones de orden más alto c_j (*crossover*), el cual crea nuevos individuos al combinar partes de (dos o más) soluciones distintas ($c_j : S \times \ldots \times S \rightarrow S$). Después de algunas generaciones el programa converge, siendo esperado que el mejor individuo represente una solución razonablemente cercana al óptimo. La figura 2.1 muestra el esquema correspondiente al algoritmo evolutivo.

Distintos sistemas evolutivos surgen a partir de esta idea del algoritmo evolutivo. Las principales diferencias entre ellos están ocultas en un nivel inferior: el uso de una estructura de datos apropiada (para la representación del cromosoma) junto con un conjunto expandido de operadores genéticos. Por

ejemplo los algoritmos genéticos clásicos usan *strings* binarios de longitud fija como un cromosoma (estructura de datos *S*) para sus individuos, y dos operadores genéticos como la mutación y el crossover binarios. En otras palabras, la estructura de un algoritmo genético es la misma que la estructura de un algoritmo evolutivo (figura 2.1) estando ocultas las diferencias en niveles inferiores. En la Programación Evolutiva los cromosomas no necesitan ser representadas por strings de bits y el proceso de alteración incluye otros operadores genéticos apropiados para la estructura y el problema dado.

```
procedimiento algoritmo evolutivo
begin
    t ← 0
    inicializar P ( t )
    evaluar P ( t )
    while (no se cumpla la condición de terminación) do
    begin
        t ← t + 1
        seleccionar P ( t ) desde P ( t – 1 )
        alterar P ( t )
        evaluar P ( t )
    end
end
```

Figura 2.1. La Estructura de un algoritmo evolutivo

Una representación natural de una solución potencial para un problema dado más una familia de operadores genéticos aplicables pueden ser bastantes útiles en la aproximación de soluciones de muchos problemas, y esta opción evolutiva es una dirección prometedora para resolver problemas en general.

4. ESTRATEGIAS EVOLUTIVAS

Las estrategias evolutivas han sido desarrolladas como un método para resolver problemas de optimización de parámetros [25, 128]; consecuentemente, un cromosoma representa a un individuo como un par de vectores,

$$\mathbf{v} = (\mathbf{x}, \boldsymbol{\sigma}).$$

Las primeras estrategias evolutivas se han basado en una población consistente de un único individuo y en un solo operador genético, la mutación. Sin embargo

la idea interesante (ausente en los algoritmos genéticos) es la representación de un individuo como un par de vectores reales, $\mathbf{v}=(\mathbf{x},\boldsymbol{\sigma})$. Donde **x** representa un punto en el espacio de búsqueda, y el segundo vector $\boldsymbol{\sigma}$ es un vector de desviaciones estándares. Las mutaciones se realizan al reemplazar **x** por:

$$\mathbf{x}^{t+1}=\mathbf{x}^{t}+N(0,\boldsymbol{\sigma}),$$

donde $N(0,\boldsymbol{\sigma})$ es un vector de números Gausianos independientes con una media igual a cero y una desviación estándar igual a $\boldsymbol{\sigma}$. (Esto concuerda con la observación biológica de que los cambios pequeños ocurren más a menudo que los grandes). El vástago (o individuo mutado) se acepta como un nuevo miembro de la población (el cual reemplaza a su padre) sí y solo sí tiene el fitness más alto y satisface todas las restricciones. De lo contrario, se elimina al hijo y la población se mantiene sin cambios. El vector estándar de desviaciones $\boldsymbol{\sigma}$ permanece sin modificaciones durante el proceso de evolución.

La idea principal detrás de estas estrategias es permitir el control de los parámetros, tal como, la varianza de la mutación; para así auto-adaptarse en lugar de modificar sus valores por algún algoritmo detérminístico [127].

Las estrategias evolutivas de mayor desarrollo son las que introduce Schwefel [124, 125, 126]:

1. la estrategia evolutiva $(\mu+\lambda)$ y,
2. la estrategia evolutiva (μ,λ).

En la estrategia $(\mu+\lambda)$, μ individuos producen λ hijos. La nueva población de $(\mu+\lambda)$ individuos es reducida por un proceso de selección de μ individuos. Mientras que la estrategia (μ,λ), los μ individuos producen λ hijos con $\lambda>\mu$, y el proceso de selección elige una nueva población de μ individuos del conjunto de λ hijos solamente. Consecuentemente, la vida de cada individuo es limitada a una generación. Esto permite que la estrategia evolutiva (μ,λ) actúe mejor en problemas con un óptimo cambiante en el tiempo, o en problemas donde la función es ruidosa (noisy).

Los operadores usados en estas dos estrategias incorporan aprendizaje en dos niveles: el parámetro de control $\boldsymbol{\sigma}$ no permanece constante, ni se modifica de

forma determinística, pero se incorpora en la estructura de individuos y sufre el proceso de evolución. Para producir un hijo, el sistema actúa en varias etapas:

1º. Selecciona dos individuos,

$(\mathbf{x}^1,\boldsymbol{\sigma}^1) = ((\mathbf{x}_1^1,.....,\mathbf{x}_n^1), (\boldsymbol{\sigma}_1^1,.....,\boldsymbol{\sigma}_n^1))$ y

$(\mathbf{x}^2, \boldsymbol{\sigma}^2)= ((\mathbf{x}_1^2,.....,\mathbf{x}_n^2), (\boldsymbol{\sigma}_1^2,.....,\boldsymbol{\sigma}_n^1))$.

Luego si $\mu > 1$ (es decir, si el tamaño de la población es mayor a uno), aplica un operador de recombinación (crossover) a los dos vectores. El crossover sobre estos grupos de información procede independientemente de cada uno de los otros.

Una variedad de mecanismos de recombinación son actualmente usados en las estrategias evolutivas y los operadores son sexuales y panmícticos. En los primeros, se recombinan dos individuos elegidos al azar desde la población de padres, donde se permite seleccionar dos veces al mismo individuo para la creación de un hijo. En cambio, la opción panmíctica de recombinación un padre elegido aleatoriamente se mantiene fijo mientras que para cada componente de sus vectores el segundo padre se selecciona al azar desde la población completa. En otras palabras, la creación de un solo hijo puede involucrar a todos los individuos padres.

Los tipos de crossovers tradicionales para las estrategias evolutivas son:

- el crossover discreto, donde el nuevo hijo es:

  $(\mathbf{x}, \boldsymbol{\sigma})= ((\mathbf{x}_1^{q_1},.....,\mathbf{x}_n^{q_n}),(\boldsymbol{\sigma}_1^{q_1},.....,\boldsymbol{\sigma}_n^{q_n}))$,

  y $\mathbf{q}_i = 1$ ó $\mathbf{q}_i = 2$ (cada componente proviene desde el primer o segundo padre preseleccionado),

- el crossover intermedio, donde el nuevo hijo es:

$(\mathbf{x}, \boldsymbol{\sigma})= (((\mathbf{x}_1^1 + \mathbf{x}_1^2)/2,....,(\mathbf{x}_n^1 + \mathbf{x}_n^2)/2),((\boldsymbol{\sigma}_1^1 + \boldsymbol{\sigma}_1^2)/2,....,(\boldsymbol{\sigma}_n^1 + \boldsymbol{\sigma}_n^2)/2))$.

Cada uno de estos operadores puede también aplicarse, en un modo global, donde el nuevo par de padres se seleccionan para *cada* componente del vector hijo.

Los operadores tradicionales pueden implementarse bajo la forma sexual o bajo la forma panmíctica.

2º. Aplica la mutación al hijo $(\mathbf{x}, \boldsymbol{\sigma})$ obtenido y el nuevo individuo resultante es $(\mathbf{x'}, \boldsymbol{\sigma'})$, donde:

$$\boldsymbol{\sigma'} = \boldsymbol{\sigma} \cdot e^{N(0,\Delta_\sigma)}$$

$$\mathbf{x'} = \mathbf{x} + N(0, \boldsymbol{\sigma'}),$$

siendo Δ_σ un parámetro del método.

Este mecanismo de mutación involucra a los propios parámetros de la estrategia evolutiva (desviación estándar y covarianzas) durante la búsqueda, explotando un enlace implícito entre un modelo interno apropiado y buenos valores de fitness. La evolución y la adaptación alcanzada de los parámetros de la estrategia de acuerdo a los requerimientos topológicos la define Schwefel como *auto-adaptación* [127].

5. PROGRAMACIÓN EVOLUTIVA

La técnica original de Programación evolutiva ha sido desarrollada por Lawrence Fogel [57]. Apunta a la evolución de la inteligencia artificial en el sentido de desarrollar la habilidad de predecir cambios en un medio ambiente. Dicho contexto ha sido descripto como una secuencia de símbolos (desde un alfabeto finito) y un supuesto algoritmo evolutivo que produce un nuevo símbolo, como salida. La salida debería maximizar la función de retribución (payoff), la cual mide la exactitud de la predicción. Por ejemplo, se puede considerar una serie de eventos, marcados por los símbolos $a_1, a_2, ...$; un algoritmo debería predecir el próximo símbolo (desconocido), $a_1, a_2, ..., a_n$. La idea de la programación evolutiva es evolucionar tales algoritmos.

Las máquinas de estados finitos han sido seleccionadas como una representación cromosomática de individuos; porque, estas máquinas proveen una representación significativa del comportamiento basado en la interpretación de símbolos. La figura 2.2 brinda un ejemplo de un diagrama de transición de una única máquina de estados finitos para un chequeo de paridad. Tal diagrama de transición presenta grafos dirigidos que contienen un nodo por cada estado, arcos que indican la transición desde un estado a otro, y valores de entrada y salida.

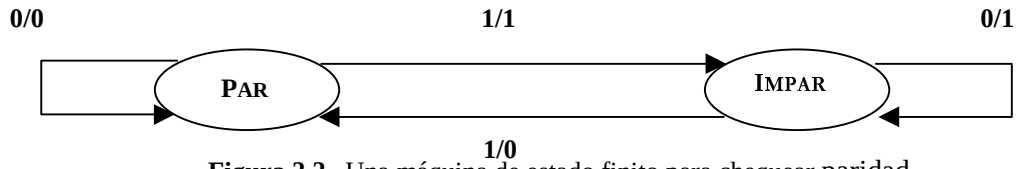

Figura 2.2. Una máquina de estado finito para chequear paridad

Aquí existen dos estados Par e Impar (la máquina inicia en el estado Par); la máquina reconoce una paridad de un string binario.

Entonces, la técnica de la programación evolutiva mantiene una población de máquinas de estados finitos, donde cada individuo representa una solución potencial al problema (por ej. representa un comportamiento en particular). Cada máquina de estado finito es evaluada para dar alguna medida de su fitness. Esto es hecho de la siguiente manera: cada máquina es expuesta al medio ambiente ya que examina todos los símbolos vistos previamente. Para cada secuencia $a_1, a_2, \ldots\ldots, a_i$ produce una salida a'_{i+1} la cual se compara con el próximo símbolo observado, a_{i+1}.

Al igual que las estrategias evolutivas, esta técnica primero crea un hijo y más tarde selecciona los individuos para la próxima generación. Cada padre produce un solo hijo. Los vástagos son creados a través de la mutación aleatoria de la población de padres (ver figura 2.3). Los operadores de mutación posibles son cinco:

1. Cambio de un símbolo de salida.
2. Cambio de una transición de estados.
3. Adición de un estado.
4. Eliminación de un estado.
5. Cambio del estado inicial.

Estas mutaciones son elegidas con respecto a alguna distribución de probabilidad (las cuales pueden cambiar durante el proceso de evolución); también es posible aplicar más de una mutación a un único padre (se toma una decisión sobre el número de mutaciones con respecto a alguna otra distribución de probabilidad).

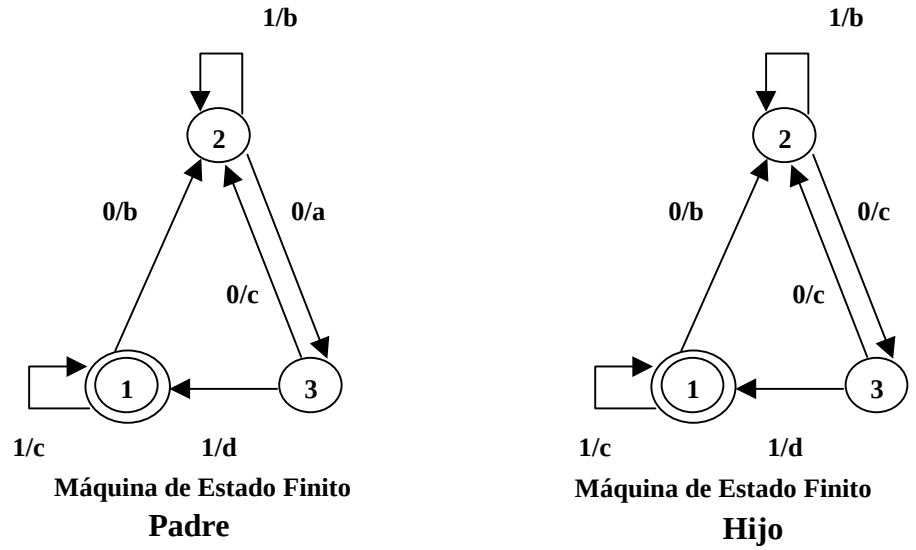

Figura 2.3. Una máquina de estado finito y su hijo. Las máquinas inician en el estado 1.

Para la próxima generación se retienen los mejores μ individuos; para poder pertenecer a la próxima generación un individuo debería estar en la cima del 50% de la población intermedia. En la versión original [57] este proceso se itera varias veces antes de que el próximo símbolo de salida esté disponible. Una vez que un nuevo símbolo resulta accesible, se agrega a la lista de símbolos conocidos, y se repite el proceso entero.

Recientemente las técnicas de la programación evolutiva han sido generalizadas para manejar problemas de optimización numéricos; para más detalles ver [52, 55].

6. PROGRAMACIÓN GENÉTICA

Koza [94, 95] recientemente ha sugerido otra interesante alternativa. Koza propone que el programa deseado debería evolucionarse a sí mismo durante el proceso de evolución. En otras palabras, en vez de resolver un problema y construir un programa evolutivo para solucionarlo, sería necesario buscar el espacio de programas computacionales posibles para resolver el problema de la mejor manera. Koza ha desarrollado una nueva tecnología, denominada Programación genética, que provee una manera de llevar a cabo tal búsqueda.

Los cinco principales pasos en el uso de la programación genética para un problema particular son:

1. Selección de terminales.
2. Selección de una función.
3. Identificación de la función de evaluación.
4. Selección de parámetros del sistema.
5. Selección de la condición de terminación.

Es importante observar que la estructura que sufre la evolución es un programa computacional estructurado jerárquicamente. El espacio de búsqueda es un hiperespacio de programas válidos, que pueden ser vistos como un espacio de árboles. Cada uno de ellos está compuesto de funciones y terminales apropiadas para el dominio de un problema en particular; el conjunto de todas las funciones y terminales se selecciona *a priori* de tal manera que alguno de estos árboles produzca una solución.

Por ejemplo, dos estructuras e_1 y e_2 (figura 2.4) representan expresiones $2x + 2.11$ y $x.\text{sen}(3.28)$, respectivamente. Un posible hijo e_3 (después del crossover entre e_1 y e_2) representa $x.\text{sen}(2x)$.

La población inicial está compuesta de tales árboles; la construcción de un árbol es sencilla. La función de evaluación asigna un valor de fitness que evalúa la performance de un árbol (programa). La evaluación se basa en un conjunto preseleccionado de casos de prueba. En general, la función de evaluación retorna la suma de distancias entre los resultados correctos y los obtenidos sobre todos los casos de prueba. La selección es proporcional; cada árbol tiene una probabilidad proporcional a su fitness de ser elegido para la próxima generación. El operador primario es un crossover que produce dos hijos desde dos padres elegidos. El crossover crea vástagos al intercambiar subárboles entre dos progenitores. Existen otros operadores, tales como: la mutación, la permutación, la edición [94]. Por ejemplo una mutación típica selecciona un nodo en un árbol y genera un nuevo subárbol en forma aleatoria, cuya raíz es el nodo elegido.

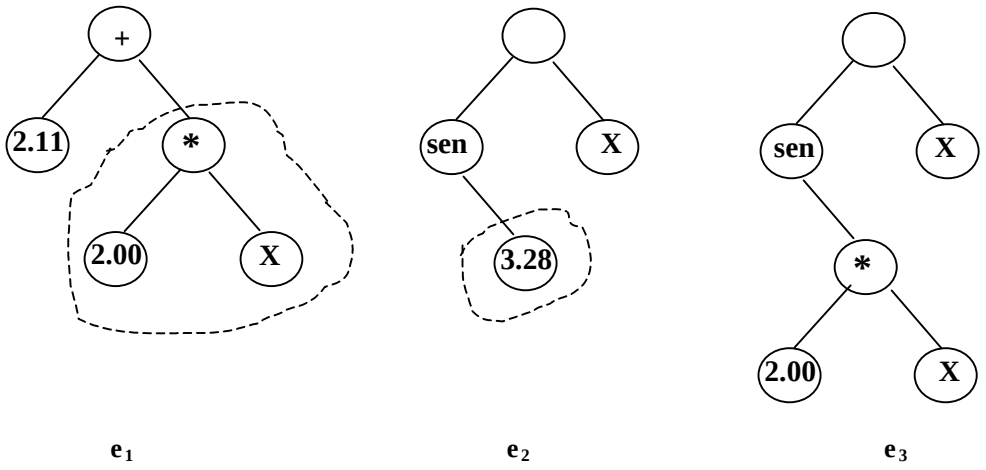

Figura 2.4. Expresión e_3: un hijo de e_1 y e_2. Las líneas de trazos partidos incluyen las áreas a ser intercambiadas durante la operación de crossover.

Koza [96] recientemente ha agregado un paso más para la construcción de un programa genético: agregar un conjunto de procedimientos. Estos procedimientos son llamados Funciones Definidas Automáticamente (ADF en inglés). Este es un concepto extremadamente útil para las técnicas de programación genética, con su mayor contribución en el área de reusabilidad de código. Las ADFs descubren y explotan las regularidades, simetrías, similitudes, patrones, y modularidades del problema en cuestión, y el programa genético final puede llamar a estos procedimientos en diferentes etapas de su ejecución.

El hecho de que la programación genética opera sobre programas de computadoras tiene pocos aspectos interesantes. Por ejemplo, los operadores pueden ser vistos como programas, que pueden sufrir una evolución por separado durante la ejecución del sistema. Adicionalmente, un conjunto de funciones pueden consistir de varios programas que realizan tareas complejas; tales funciones pueden desarrollarse, además, durante la ejecución evolutiva (ej. ADF). Actualmente, esta es una de las áreas de desarrollo de mayor interés en la computación evolutiva y ha acumulado una importante cantidad de datos experimentales (ver por ej. [95, 96]).