Capítulo III

ALGORITMOS EVOLUTIVOS

3.1. INTRODUCCIÓN

Ha habido un interés creciente en algoritmos que se basan en el principio de evolución (supervivencia del mejor). Un término común, aceptado recientemente, para referirse a tales técnicas es *algoritmos evolutivos* (EA) o métodos de *computación evolutiva* (EC).

En general, cualquier tarea abstracta a ser realizada se puede pensar como la resolución de un problema, el cual, a su vez, se puede percibir como una búsqueda a través de un espacio de soluciones potenciales [24]. Como usualmente se busca la mejor solución, se puede pensar en esta tarea como un proceso de optimización. Para espacios pequeños, generalmente es suficiente usar los métodos exhaustivos clásicos; pero para espacios grandes se deben emplear técnicas especiales de inteligencia artificial. Los algoritmos evolutivos caen dentro de esas técnicas; son algoritmos estocásticos cuyos métodos de búsqueda modelan un fenómeno natural: la herencia genética y la rivalidad darwiniana para la supervivencia. Como se indica en [28]: "*la metáfora subyacente en algoritmos genéticos es la evolución natural. En evolución, el problema de cada especie es un problema de búsqueda de adaptaciones benéficas en ambientes cambiantes y complicados. El conocimiento que cada especie ha ganado se embebe en el cromosoma de sus miembros*".

Hay varias variantes de algoritmos evolutivos, las cuales incluyen:

- ✓ *programación evolutiva*: se hace evolucionar una población de máquinas de estados finitos sometiéndolas a transformaciones unitarias.
- ✓ *estrategias evolutivas*: se hace evolucionar una población de estructuras compuestas por un vector real y una variable aleatoria (parámetro) que codifican las posibles soluciones de un problema numérico y las dimensiones de los cambios o saltos. La selección es implícita.

- ✓ *programación genética*: se hace evolucionar una población de estructuras de datos sometiéndolas a una serie de transformaciones específicas y a un proceso de selección.
- ✓ *algoritmos genéticos*: se hace evolucionar una población de enteros binarios sometiéndolos a transformaciones unitarias y binarias genéticas y a un proceso de selección.

Hay también muchos sistemas híbridos los cuales incorporan varias características de los paradigmas antes mencionados y, consecuentemente, es difícil su clasificación; de todas maneras, se referirán a ellos como métodos de computación evolutiva. La estructura de cualquier algoritmo evolutivo es bastante similar; una estructura simple es la que se muestra en la figura 3.1 [24].

Los algoritmos evolutivos mantienen una población de individuos, $P(t) = \{x^t_1, x^t_2, \ldots, x^t_n\}$ en la iteración t, donde n es el tamaño de la población (*pop_size*). Cada individuo representa una solución potencial al problema, y se implementa como una estructura de datos *S*. Cada solución x^t_i se evalúa para dar alguna medida de su "fitness". Entonces, una nueva población (iteración $t + 1$) se forma al seleccionar los individuos más idóneos (paso seleccionar). Algunos miembros de la nueva población se someten a transformaciones (paso alterar) por medio de operadores

```
procedure: algoritmo evolutivo
  begin
    t ← 0;
    iniciar P(t);
    evaluar P(t);
    while no(condición de terminación) do
        t ← t + 1;
        seleccionar P(t) de P(t-1);
        alterar P(t);
        evaluar P(t);
    end while
  end procedure
```

Figura 3.1 Estructura de un algoritmo evolutivo.

"genéticos" para formar nuevas soluciones. Hay transformaciones unarias m_i (tipo mutación), las cuales crean nuevos individuos al producir pequeños cambios en un único individuo ($m_i : S \rightarrow S$), y transformaciones de orden superior c_j (tipo crossover), la cuales crean nuevos individuos al combinar varias partes de dos o más individuos ($m_i : S \times \ldots \times S \rightarrow S$). En la mayoría de los casos el crossover involucra sólo dos padres, sin embargo, no es necesariamente el caso. En [38] se ha investigado los méritos de "orgía", donde se involucran más de dos padres en el proceso de reproducción. Luego de algún número de generaciones, el algoritmo converge; es de esperar que el mejor individuo represente una solución cercana a la óptima.

A pesar de las similitudes entre los algoritmos evolutivos, hay también muchas diferencias entre ellos (a menudo ocultas a un bajo nivel de abstracción). Frecuentemente usan estructuras de datos *S* diferentes para la representación de los cromosomas, en consecuencia, los operadores genéticos también son diferentes. Pueden incorporar o no alguna otra información (para controlar el proceso de búsqueda) en sus genes. Hay también otras diferencias; por ejemplo, las dos líneas de la figura 1:

seleccionar *P(t)* de *P(t-1)*;

alterar *P(t)*;

pueden aparecer en orden inverso: en estrategias evolutivas primero se altera la población y luego se forma la nueva población por el proceso de selección. Hay muchos métodos para seleccionar individuos para supervivencia y reproducción. Esos métodos incluyen:

- ✓ *selección proporcional*: la probabilidad de selección es proporcional al fitness del individuo,
- ✓ *métodos de ranking*: todos los individuos de la población se ordenan de mejor a peor y las probabilidades de selección son fijas durante todo el proceso de evolución, y
- ✓ *selección por torneo*: algunos individuos (usualmente dos) compiten por ser seleccionados a la nueva generación; este paso de competencia (torneo) se repite *pop_size* veces.

La selección proporcional puede necesitar el uso de métodos de truncamiento o de escalamiento, hay diferentes formas para asignar probabilidades en métodos de ranking (distribuciones lineales o no lineales), el tamaño de un torneo juega un rol significativo en el método de selección por torneo.

Es importante determinar las políticas generacionales, es decir especificar la forma en la cual quedará conformada la próxima generación. Por ejemplo, es posible reemplazar la población entera por la población de hijos, o seleccionar los mejores individuos de las dos poblaciones (padres e hijos), esta selección se puede hacer de una manera determinística o no determinística. Es también posible producir pocos hijos (en particular, uno único), con los cuales reemplazar algunos individuos (sistemas basados en tales políticas generacionales son llamados “steady state”). En un modelo generacional el conjunto de los padres se mantiene fijo mientras se generan todos los hijos que formarán parte de la próxima generación. También, se puede usar un modelo elitista, el cual pasa el mejor individuo de una generación a otra, esto significa que si el mejor individuo de la generación actual corre el riesgo de perderse, ya sea por la selección o los operadores genéticos, el sistema fuerza a que quede en la próxima generación; tal modelo es muy útil para resolver muchas clases de problemas de optimización.

Las representaciones (estructuras de datos necesarias implementar un individuo en el espacio fenotípico) de las soluciones potenciales para un problema particular, junto con el conjunto de operadores genéticos, constituyen los componentes esenciales de cualquier algoritmo evolutivo. Esos son los elementos claves que permiten distinguir varios paradigmas de métodos evolutivos.

Los algoritmos evolutivos han recibido mucha atención tanto desde el punto de vista académico como industrial. Las herramientas basadas en EA tienen un impacto creciente en compañías (prediciendo el mercado financiero), en fábricas (scheduling) debido a su poder de búsqueda, optimización, adaptación y aprendizaje. Una de las áreas recientes de aplicación para esas técnicas es el campo de la ingeniería industrial, donde se incluye scheduling y ordenamiento en sistemas de manufactura, diseño asistido por computadora, facilidad de layout y problemas de ubicación, y muchos más. Se recurren a los algoritmos genéticos por su simplicidad, y facilidad de extensión.

Aunque los algoritmos evolutivos han sido exitosamente aplicados a muchos problemas prácticos y el número de aplicaciones se incrementa, también se presentan varias desventajas. Hay poco conocimiento respecto de que características del dominio los hacen apropiados o inapropiados. Se han sugerido varias modificaciones para mitigar las dificultades tanto en el manejo de la información codificada como en las

formas de representar el espacio del problema. Tres importantes afirmaciones se han hecho respecto de porque los algoritmos evolutivos trabajan bien:

- ✓ Las poblaciones de soluciones candidatas proveen muestreos independientes.
- ✓ La selección es un mecanismo que preserva buenas soluciones.
- ✓ Las soluciones parciales se pueden modificar eficientemente y combinar a través de varios operadores genéticos.

3.2. ALGORITMOS GENÉTICOS

La forma usual de un algoritmo genético (GA) fue descripta por Goldberg [75]. Los algoritmos genéticos (GAs) son técnicas de búsqueda estocásticas basadas en los mecanismos de selección y genética natural. Los GAs, a diferencia de las técnicas de búsqueda convencionales, comienzan con un conjunto inicial de soluciones, generado en forma aleatoria, llamado *población*. Cada individuo en la población se llama *cromosoma* y representa una solución al problema. Un cromosoma es un string de símbolos y generalmente, pero no necesariamente, un string de bits binario de longitud fijo. Los cromosomas evolucionan a través de sucesivas iteraciones, llamadas *generaciones*. Durante cada generación, los cromosomas se *evalúan*, usando alguna medida de *fitness* (aptitud) [62]. Para crear la próxima generación, se forman nuevos cromosomas, llamados *hijos*, ya sea:

- ✓ mezclando dos cromosomas de la generación actual usando un operador de crossover, ó,
- ✓ modificando un cromosoma por medio de un operador de mutación.

Una nueva generación se forma al seleccionar, acorde a los valores de fitness, alguno de los padres e hijos, y al rechazar algunos, a fin de mantener constante el tamaño de la población.

Los cromosomas mejor adaptados tienen alta probabilidad de ser seleccionados. Luego de varias generaciones, el algoritmo converge al mejor cromosoma, el cual se espera que represente una solución óptima o subóptima.

Los algoritmos genéticos tienen varias diferencias con los procedimientos convencionales de búsqueda y optimización. Goldberg [75] las ha resumido como sigue:

- ✓ Los algoritmos genéticos trabajan con un conjunto de soluciones codificadas, no con la solución en sí misma.
- ✓ Los algoritmos genéticos buscan en una población de soluciones, no con una única solución.
- ✓ Los algoritmos genéticos usan información de retribución (función de fitness), no necesitan conocimiento derivado o auxiliar.
- ✓ Los algoritmos genéticos usan reglas de transición probabilísticas, no reglas determinísticas.

Los GAs han recibido considerable atención como técnica de optimización en función de sus potenciales. Hay tres ventajas principales cuando se aplican GA para la optimización de problemas:

- ✓ Los GAs no tienen muchos requerimientos matemáticos de los problemas de optimización. Dada la naturaleza evolutiva, los GAs buscarán soluciones sin considerar el funcionamiento interno del problema. Un GA puede manejar cualquier clase de funciones objetivo y cualquier clase de restricciones (lineales o no lineales) definidas sobre espacios de búsqueda discretos, continuos o una mezcla de ellos.
- ✓ La naturaleza de los operadores de evolución hacen a los GAs muy efectivos para realizar búsquedas globales. Las opciones tradicionales realizan una búsqueda local con un procedimiento de convergencia, el cual compara los valores de puntos cercanos y se mueve a puntos relativamente óptimos.
- ✓ Los GAs proveen una gran flexibilidad para realizar híbridos con heurísticas dependientes del dominio, a fin de lograr una implementación efectiva para problemas específicos.

3.3. ESTRATEGIAS EVOLUTIVAS

Las estrategias evolutivas (ES) se desarrollaron como un método para resolver problemas de optimización paramétrica [127]; consecuentemente, un cromosoma representa un individuo como un par de vectores de valor flotante.

Las primeras estrategias evolutivas se basaban sobre una población de un único individuo. Se usaba un único operador genético en el proceso de evolución: la mutación. Sin embargo, la idea interesante fue representar un individuo como un par de vectores de valor flotante, es decir $v = (\mathbf{x}, \boldsymbol{\sigma})$. Aquí, el primer vector **x** representa un punto en el

espacio; el segundo vector σ es un vector de desviaciones estándares. La mutación se realizaba 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 random gausianos independientes con una media de cero y una desviación estándar $\boldsymbol{\sigma}$ (esto está en armonía con la observación biológica que los cambios pequeños ocurren más frecuentemente que los grandes). El hijo (el individuo mutado) se acepta como un nuevo miembro de la población (reemplaza a su padre) si y solo si tiene fitness mejor y satisface todas las restricciones (si es que las hay). Por ejemplo, si f es la función objetivo sin restricciones a maximizar, un hijo $(\mathbf{x}^{t+1}, \boldsymbol{\sigma})$ reemplaza al padre $(\mathbf{x}^{t}, \boldsymbol{\sigma})$ si y solo si $f(\mathbf{x}^{t+1}) > f(\mathbf{x}^{t})$. En caso contrario, se elimina al hijo y la población permanece sin cambio.

El vector de desviaciones estándares σ permanece sin cambio durante el proceso de evolución. Si todos los componentes de este vector son idénticos, es decir, $\boldsymbol{\sigma} = (\sigma,\ldots,\sigma)$, y el problema de optimización es regular, es posible demostrar el teorema de convergencia [7]:

Teorema de Convergencia: para $\sigma > 0$ y un problema de optimización regular con $f_{opt} > -\infty$ (minimización) o $f_{opt} < \infty$ (maximización), se tiene

$$p\{\lim_{t\to\infty} f(x^{t}) = f_{opt}\} = 1$$

Las estrategias evolutivas evolucionaron más tarde [127] para madurar como:

$$(\mu+\lambda)\text{-ES y } (\mu,\lambda)\text{-ES}$$

la principal idea detrás de estas estrategias fue permitir el control de parámetros (como la varianza de mutación) para auto adaptación más que para cambiar sus valores por algún algoritmo determinístico.

En (μ+λ)-ES, μ individuos producen λ hijos. La nueva población de (μ+λ) individuos se reduce, por un proceso de selección, nuevamente a μ individuos. Por otra parte, en (μ,λ)-ES, los μ individuos producen λ hijos (λ>μ) y el proceso de selección elige una nueva población de μ individuos desde el conjunto de λ hijos. Al hacer esto, la vida de cada individuo se limita a una generación. Esto permite que (μ,λ)-ES trabaje mejor sobre problemas con un óptimo moviéndose sobre el tiempo, o en problemas donde la función objetivo presenta mucho ruido.

La selección (μ+λ), con la supervivencia garantizada de los mejores individuos, parece ser más efectiva, porque un curso monótono de evolución se realiza de esta

forma. Sin embargo, este mecanismo de selección tiene varias desventajas cuando se compara con la selección (μ,λ), la cual restringe el tiempo de vida de los individuos a una generación:

- ✓ En el caso de medio ambientes cambiantes la selección $(\mu+\lambda)$ preserva la solución y no es capaz de seguir un óptimo en movimiento.
- ✓ La capacidad de la selección (μ,λ) para olvidar buenas soluciones (en principio permite dejar óptimos locales pequeños) es ventajosa en el caso de topologías multimodales.
- ✓ La selección $(\mu+\lambda)$ obstaculiza el mecanismo de auto adaptación para que trabaje adecuadamente con respecto a los parámetros estratégicos, porque los parámetros estratégicos mal adaptados pueden sobrevivir por un gran número de generaciones cuando producen mejoras en el fitness.

Por consiguiente, se recomienda la selección (μ,λ). Resumiendo las condiciones para una exitosa auto adaptación de parámetros de estrategia, se obtiene la siguiente lista [6]:

- ✓ Se necesita la estrategia (μ,λ) a fin de facilitar la extinción de individuos mal adaptados.
- ✓ La presión selectiva puede no volverse muy fuerte.
- ✓ Es necesaria la recombinación de los parámetros estratégicos (generalmente, la recombinación intermedia da buenos resultados).

El operador usado en $(\mu+\lambda)$-ES y (μ,λ)-ES incorpora dos niveles de aprendizaje: su parámetro de control $\boldsymbol{\sigma}$ no es más constante, no se cambia por algún algoritmo determinístico (como la regla de éxito 1/5), pero se incorpora en la estructura del individuo y se somete al proceso de evolución. Para producir un hijo, el sistema actúa en varias etapas:

- ✓ Seleccionar dos individuos:

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

  $(\mathbf{x}^2, \boldsymbol{\sigma}^2) = (x^2_2, \ldots, x^2_n), (\sigma^2_2,\ldots, \sigma^2_n)$.
- ✓ Aplicar un operador de recombinación (crossover). En ES, la recombinación se usa para la creación de todos los hijos, cuando $\mu > 1$. Tanto las variables objeto como los parámetros estratégicos están sujetos a recombinación y el operador de recombinación puede ser diferente para variables objeto y desviaciones estándar. Esto implica que la recombinación de esos grupos de información se realiza en

forma independiente unos de otros, es decir, no se restringe que los parámetros estratégicos se originen desde los mismos padres que los valores objeto. Los operadores de recombinación tradicional de ES se denominan :

✓ *discretos,* donde el nuevo hijo es

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

donde q_i=1 o q_i=2 (así cada componente proviene del primer o del segundo padre preseleccionados). Por cada componente del vector se decide en forma aleatoria desde cuál de los dos padres se copia el componente al hijo.

✓ *intermedios,* donde el nuevo hijo es

$(\mathbf{x}, \boldsymbol{\sigma}) = ((x^1_1 + x^2_2)/2, \ldots, (x^1_n + x^2_n)/2), ((\sigma^1_1 + \sigma^2_2)/2, \ldots, (\sigma^1_n + \sigma^2_n)/2)$

indica que los componentes de los hijos se obtienen al calcular la media aritmética de los componentes correspondientes de ambos padres.

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

✓ Aplicar mutación al hijo $(\mathbf{x}, \boldsymbol{\sigma})$ obtenido; el nuevo hijo resultante es $(\mathbf{x}', \boldsymbol{\sigma}')$, donde

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

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

donde $\Delta\boldsymbol{\sigma}$ es un parámetro del método.

3.4. PROGRAMACIÓN EVOLUTIVA

La técnica original de programación evolutiva (EP) se desarrolló por Fogel [64]. Se proponía la evolución de inteligencia artificial en el sentido de desarrollar la habilidad para predecir cambios en un medio ambiente. El medio ambiente se describió como una secuencia de símbolos (desde un alfabeto finito) y el algoritmo suponía la producción por evolución, como una salida, de un nuevo símbolo. El símbolo de salida maximiza la función de retribución (payoff), la cual mide la exactitud de la predicción.

Por ejemplo, se puede considerar una serie de eventos, marcada por símbolos $a_1, a_2, \ldots, a_n$; un algoritmo debería predecir el próximo símbolo (no conocido), es decir a_{n+1} sobre las bases de los símbolos ya conocidos $a_1, a_2, \ldots, a_n$. La idea de programación evolutiva fue evolucionar tal algoritmo.

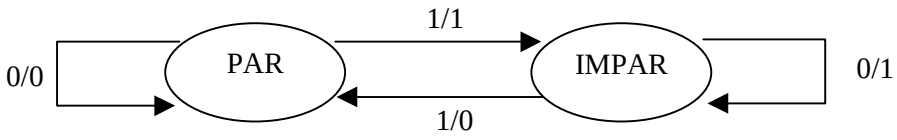

Figura 3.2 Una FSM para un chequeo de paridad.

Se seleccionaron máquinas de estado finito (FSM) para la representación del cromosoma de los individuos; luego, todas las máquinas de estado finito proveen una representación significativa del comportamiento basado sobre interpretación de símbolos. La figura 3.2 [24] presenta un ejemplo de un diagrama de transición de una máquina de estado finito simple para un chequeo de paridad. Tal diagrama de transición es un grafo dirigido que contiene un nodo en cada etapa y ejes que indican la transición desde un estado a otro, valores de entrada y salida (notación: *a*/*b* cercano a un arco que une el estado S_1 con el estado S_2, sugiere que el valor de entrada *a*, mientras la máquina está en el estado S_1, resulta en un valor de salida *b* y en el próximo estado S_2).

Hay dos estados PAR e IMPAR (la máquina comienza en el estado PAR); la máquina reconoce la paridad de un string binario.

La técnica de programación evolutiva mantiene una población de máquinas de estado finito (FMS); cada individuo representa una solución potencial al problema (representa un comportamiento particular). Cada FSM se evalúa para dar alguna medida de su fitness. Esto se hace de la siguiente forma: cada FSM está expuesta al ambiente en el sentido de que examina todos los símbolos vistos previamente. Por cada subsecuencia, $a_1, a_2, \ldots, a_i$ produce una salida a'_{i+1}, la cual se compara con el próximo símbolo observado, a_{i+1}. Por ejemplo, si se han visto *n* símbolos, un FSM hace *n* predicciones (una por cada uno de los substrings a_1, a_1, a_2 y así sucesivamente hasta $a_1, a_2, \ldots, a_n$); la función de fitness toma en cuenta la performance total.

Como en estrategias evolutivas, la técnica de programación evolutiva primero crea hijos y luego selecciona individuos para la próxima generación. Cada padre produce un único hijo; así se duplica el tamaño de la población intermedia (como en (*pop_size*, *pop_size*)-ES). Los hijos (una nueva FSM) se crean por mutaciones aleatorias de la población de padres. Hay cinco posibles operadores de mutación: cambio de un símbolo de salida, cambio de una transición de estado, adición de un estado, eliminación de un estado, y cambio de un estado inicial (hay algunas restricciones

iniciales sobre el número máximo o mínimo de estados). Esas mutaciones se eligen con respecto a alguna distribución de probabilidades (la cual puede cambiar durante el proceso de evolución); también es posible aplicar más de una mutación a un único padre (la decisión sobre el número de mutaciones de un individuo particular se hace con respecto a alguna otra distribución de probabilidades).

Los mejores *pop_size* individuos se retienen para la próxima generación, es decir, para calificar para la próxima generación, un individuo deberá estar por el encima del 50% de la población intermedia. En la versión original [64], este proceso se iteraba varias veces antes de que estuviera disponible la próxima salida de símbolos. Una vez que se encuentra disponible un nuevo símbolo, se adicionaba a la lista de símbolos conocidos, y se repite el proceso total.

Desde luego, el proceso detallado previamente se puede extender de muchas maneras; como se describe en [58].

Las técnicas de programación evolutiva fueron generalizadas para tratar con problemas de optimización numéricas [55, 58]. En [64, 56, 60, 61, 128, 100] se encuentran ejemplos de técnicas de programación evolutiva.

3.5. PROGRAMACIÓN GENÉTICA

Fue desarrollada por Koza [88, 89]. Koza sugiere que el programa deseado deberá evolucionar durante el proceso de evolución. En otras palabras, en lugar de resolver un problema, y de construir un programa evolutivo para resolver el problema, se deberá buscar en el espacio de posibles programas de computación el que mejor se adecue. Koza desarrolló una nueva metodología, llamada programación genética (GP), la cual provee una forma para realizar tal búsqueda.

Hay cinco pasos para usar programación genética en un problema particular. Ellos son:

- ✓ Selección de terminales.
- ✓ Selección de una función.
- ✓ Identificación de la función de evaluación.
- ✓ Selección de los parámetros del sistema.
- ✓ Selección de la condición de terminación.

La estructura que pasa por el proceso de evolución es la estructura jerárquica del programa de computación. El espacio de búsqueda es un hiper-espacio de programas válidos, los cuáles se pueden ver como un espacio de árboles. Cada árbol está compuesto de funciones y terminales apropiadas para el dominio de un problema particular; el conjunto de todas las funciones y terminales se selecciona a priori de tal forma que algunos de los árboles formados lleve a una solución.

Por ejemplo, dos estructuras e_1 y e_2 (figura 3.3) representan expresiones $2x + 2.11$ y $x \cdot \text{sen}(3.2)$, respectivamente. Un posible hijo e_3 (luego del crossover de e_1 y e_2) representa $x \cdot \text{sen}(2x)$ [24].

La población inicial está compuesta de tales árboles; la construcción de un árbol en forma aleatoria es sencilla. La función de evaluación asigna un valor de fitness el cual evalúa la performance de un árbol (programa). La evaluación se basa sobre un conjunto preseleccionado de casos de testeo; en general, la función de evaluación retorna la suma de distancias entre el resultado correcto y el obtenido sobre todos los casos de prueba.

La selección es proporcional; cada árbol tiene una probabilidad de ser seleccionado para la próxima generación proporcionalmente a su fitness. La operación primaria es el crossover que produce dos hijos desde dos padres seleccionados. El crossover crea hijos al intercambiar subárboles entre dos padres. Hay otros operadores como mutación, permutación, edición, etc. Por ejemplo, una mutación típica selecciona un nodo en un árbol y genera un nuevo subárbol el cual se origina en el nodo seleccionado.

Hay cinco pasos para construir un programa genético para un problema particular. Koza [91] recientemente considera la ventaja de agregar una característica adicional: un conjunto de procedimientos. Esos procedimientos se llaman funciones definidas

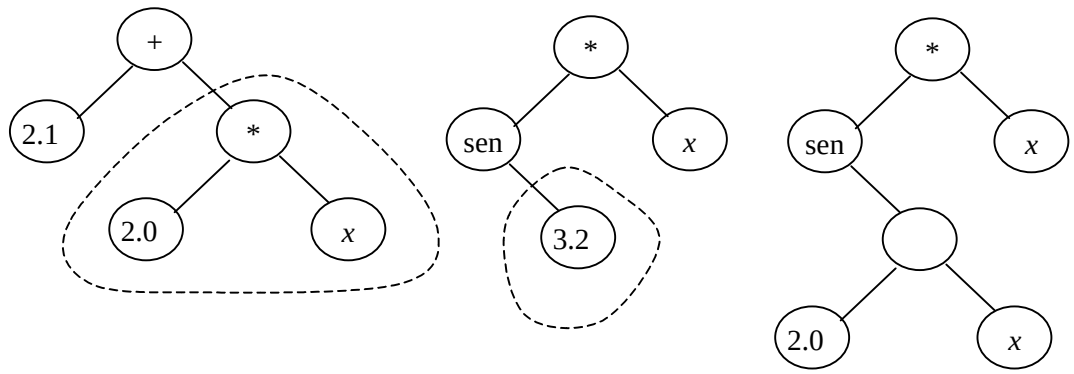

Figura 3.3 Expresión e_3: un hijo de e_1 y e_2. La línea con trazo cortado incluye área intercambiadas durante la operación de crossover.

automáticamente (ADF). Este parece ser un concepto extremadamente útil para técnicas de programación genética, con su mejor contribución al área de reusabilidad de código. ADF descubre y explota las regularidades, simetrías, similitudes, patrones, y modularidades del problema y el programa genético puede invocar a esos procedimientos en diferentes momentos de su ejecución.

El hecho de que los programas genéticos trabajen sobre programas de computación tiene aspectos muy interesantes. Por ejemplo, los operadores se pueden ver como programas, los cuales se pueden someter a una evolución separada durante la ejecución del sistema. Adicionalmente, un conjunto de funciones pueden consistir de varios programas, los cuales realizan tareas complejas; tales funciones pueden evolucionar durante la ejecución evolutiva. En general, un GP tiende a hacer un trabajo bastante bueno para hallar una solución general siempre que:

- ✓ Se le brinden casos de entrenamientos para ser capaz de deducir la solución general.
- ✓ Que la función y el conjunto de terminales hayan sido elegidos a fin de influir apropiadamente en el espacio de soluciones para un problema dado.

Es una de las áreas de desarrollo actual en el campo de la computación evolutiva [89, 91, 87, 3].

3.6. OTRAS TÉCNICAS

Los algoritmos evolutivos han sido modificados al agregar en el algoritmo conocimiento específico del problema. Varios trabajos han discutido técnicas de inicio, diferentes representaciones, técnicas de decodificación (traslación desde representaciones genéticas a representaciones fenotípicas), y el uso de heurísticas para operadores genéticos.

Tales sistemas no estándares o híbridos tienen significativa popularidad en la comunidad de computación evolutiva. Frecuentemente esos sistemas, extendidos por el conocimiento específico del problema, mejoran tanto a otros métodos evolutivos clásicos como a otras técnicas estándares [101, 102]. Por ejemplo, un sistema Genético-2N [101] construido para el problema de transporte no lineal usa una representación matricial para sus cromosomas, una mutación específica del problema (principal operador, con probabilidad 0.4) y crossover aritmético (operador de background, usado con probabilidad 0.05). Es difícil clasificar tal sistema: no es

realmente un algoritmo genético, ya que puede ejecutar sólo con el operador de mutación sin un significativo decremento de la calidad de los resultados. Sin embargo, todas las entradas en la matriz son valores de punto flotantes. No es una estrategia evolutiva, ya que no codifica ningún parámetro de control en la estructura del cromosoma. Claramente, no cae dentro de la programación genética ni de la programación evolutiva. Es una técnica de computación evolutiva apuntada a la resolución de un problema particular.

Hay pocas heurísticas para guiar al usuario en la selección de estructuras de datos y operadores apropiados para un problema particular. Es de conocimiento general que para un problema de optimización numérico se deberá usar una estrategia evolutiva o un algoritmo genético con representación de punto flotante, mientras algunas versiones de algoritmos genéticos deberán ser las mejores para controlar problemas de optimización combinatoria. Los programas genéticos son buenos para descubrir reglas dadas como un programa de computación, y las técnicas de programación evolutivas se pueden usar exitosamente para modelar el comportamiento de un sistema.

3.7. DIFERENCIAS ENTRE LAS DISTINTAS TÉCNICAS DE EA

Aunque todos los algoritmos comparten un meta-modelo, cada uno enfatiza diferentes características como las más importantes para el modelado exitoso del proceso evolutivo. Dejando de lado las cuestiones de representación, las diferencias más notorias se presentan en la interpretación del rol de los operadores genéticos. En programación evolutiva, no se usa recombinación, mientras que en algoritmos genéticos juega un rol dominante, abandonando la influencia de la mutación casi completamente. Las estrategias evolutivas usan ambos operadores, indicando un necesario uso de recombinación sobre los parámetros estratégicos para lograr una auto adaptación exitosa.

Tanto la programación evolutiva como los algoritmos genéticos insisten en la selección probabilística, aunque con diferencias en la implementación; el primer caso se refiere a la característica de selección en la naturaleza y en el segundo caso a la analogía con el multiarmed bandit (bandido de múltiples armas) [6]. En contraste, las estrategias evolutivas usan un mecanismo de selección estrictamente determinístico, siendo

extintivo en el sentido de excluir definitivamente de la selección a los peores individuos.

Mientras en programación evolutiva algunos individuos son excluidos de la selección, los algoritmos genéticos hacen uso de *preservación* en el sentido de que cada individuo recibe una probabilidad de selección distinta de cero. La propiedad elitista es implícita en la selección en programación evolutiva, mientras esto ocurre explícitamente en estrategias evolutivas ($(\mu+\lambda)$-ES) y algoritmos genéticos.