Universidad Autónoma de Guerrero

Facultad de matemáticas

Exponenciaciones Modulares Usando Algoritmos Genéticos Con Representación Entera

Tesis para obtener el grado de Lic. En Matemáticas área en Computación

Presenta: Raúl Juárez Morales

Asesor: Dr. Carlos Artemio Coello Coello

Coasesor: Dr. José Torres Jiménez

13 de octubre de 2006

Agradecimientos

A mi madre y a mi padre por los valores que me han inculcado y por llevarme por un buen camino por apoyarme durante toda mi carrera.

A mis amigos de siempre (Enrique, Roberto, Arturo, Luís Daniel, José Ángel).

A Aleyda Locena Hernández por comprenderme y ayudarme cuando tengo muchos problemas además por ser la persona que me inspira para seguir adelante.

Al Dr. Carlos Coello Coello y al Dr. José Torres Jiménes por permitirme un poco de su tiempo, y por haberme asesorado.

A Virgilio Cruz Guzmán por guiarme durante la licenciatura y por resolver mis dudas en la programación.

Dedicatorias

A mi madre Carmela Morales Gonzales y a mi padre Ramiro Juárez Cruz.

A mi novia Aleyda

A mis hermanos y amigos

Índice

Índice de figuras

Índice de Tablas

Capítulo I

Introducción

1. Introducción

A inicios del siglo anterior, con el desarrollo de las computadoras, un gran número de problemas que hasta entonces no eran tratables a mano, pudieron ser estudiados más concretamente.

Con la aparición de la computación evolutiva muchos de estos problemas han sido resueltos gracias a los algoritmos genéticos, estrategias evolutivas y programación evolutiva.

El algoritmo genético (**AG**) es una técnica de búsqueda basada en la teoría de la evolución de Darwin, que ha cobrado tremenda popularidad alrededor del mundo durante los últimos años. Se presentarán aquí los conceptos básicos que se requieren para abordarla.

Un algoritmo genético (**AG**), permite que las computadoras solucionen problemas difíciles mediante el uso del principio de selección natural (o supervivencia del más apto).

Búsqueda, optimización y aprendizaje son los problemas a los que se pueden aplicar los algoritmos evolutivos, en cualquiera de sus formas.

En realidad, los algoritmos de búsqueda abarcan prácticamente todo algoritmo para resolver problemas automáticamente. Habitualmente, en Informática se habla de búsqueda cuando hay que hallar información, siguiendo un determinado criterio, dentro de un conjunto de datos almacenados; sin embargo, aquí nos referiremos a otro tipo de algoritmos de búsqueda, a saber, aquellos que, dado el espacio de todas las posibles soluciones a un problema, y partiendo de una solución inicial, son capaces de encontrar la mejor solución posible. Este tipo de algoritmos son catalogados como algoritmos

aproximados.

En este documento el capítulo 2, describe lo que es la computación evolutiva. En dicho capítulo se enmarcan los antecedentes de la computación evolutiva y sus principales paradigmas, v.g. dar una breve explicación de cuáles son los principios de los AG, es decir cuál es el sustento teórico de los AG.

En el capítulo 3 se detalla lo que son los AG, describiendo sus caracteristicas, cómo debemos de construir su población, y los tipos de operadores de selección que se pueden usar al implementar un AG. Se describe en detalle el operador de cruza, ya que éste es el operador principal de un AG. Este capítulo finaliza con una descripción del operador de mutación de un AG.

En el capítulo 4 se da un bosquejo del problema que se aborda en esta tesis (Exponenciaciones Modulares), indicándose cuál es la importancia de este tema, y como podemos utilizar esta teoría en la vida real.

En el capítulo 5 se describe la solución propuesta al problema. Dicha propuesta se basa en el uso de un AG que ayuda a optimizar las expoenciaciones modulares. En este capítulo se explica a detalle cómo se fue construyendo el AG en cada una de sus fases (Población inicial, Selección, Cruza y Mutación).

El capítulo 6 presenta una serie de resultados generados por el AG presentado en el capítulo 5. Estos resultados evidencian la efectividad del AG en el problema de estudio. Como parte del análisis de resultados se hace una comparativa con los resultados producidos por otros algoritmos.

Para finalizar, en el capítulo 7 se presentan las conclusiones de este trabajo, así como algunas de las posibles líneas futuras de investigación para

los que pudieran interesarse en continuar trabajando en este tema.

Capítulo II

Computación Evolutiva

2. Computación Evolutiva

La computación evolutiva se refiere a un conjunto de técnicas de búsqueda y optimización inspiradas en el proceso de selección natural (la supervivencia del más apto) que reina en las especies del planeta [4].

2.1. Introducción

La computación evolutiva es una de las heurísticas más usadas para resolver problemas no lineales de optimización, El uso de técnicas heurísticas tales como los algoritmos evolutivos se justifica cuando no es garantizar que se hallará de manera determinista el óptimo global de un problema, que es precisamente el caso cuando lidiamos con el problema general de optimización no lineal. En estos casos, la computación evolutiva es una buena opción, dado que suele obtener buenas aproximaciones del óptimo global de diversos tipos de problemas de optimización. Sin embargo, dada su naturaleza heurística, la computación evolutiva no puede garantizar que las soluciones que encuentra son óptimas.

2.2. Antecedentes

Esta sección trata de explicar cómo se originó el modelo en el que se basa la computación evolutiva, analizando las ideas fundamentales del seleccionismo Weismann, la genética de Mendel y la teoría evolutiva de Darwin. Fueron esta tres ideas fundamentales las que dieron pie al Neo-Darwinismo.

El Creacionismo dice que Dios creó el cielo y la Tierra así como todas

las especies que en ella habitan, cada una por separado. Por muchos años, el creacionismo fue la explicación más aceptada sobre el origen de las especies. Sin embárgo, el descontento de algunos científicos con creacionismo dio lugar al desarrollo de los principios que fungen como los orígenes de la Computación Evolutiva. A mediados del siglo XIX el zoólogo francés Jean-Baptiste Lamarck creó su propia teoría de la evolución en la que argumentaba que las características adquiridas por un organismo como resultado de un proceso de adaptación a lo largo de su vida son heredadas a las siguientes generaciones [2].

El científico alemán August Weismann demostró que la teoría de Lamarck estaba equivocada: cortó la cola de un grupo de ratas por varias generaciones y observó que la longitud de la cola de las nuevas generaciones no se veía afectada. Para Weismann, la selección natural era lo único capaz de alterar la composición genética de un organismo [3].

En 1859, Charles Darwin publicó un libro titulado: El origen de las especies [4] en el que argumentó que la similitud entre padres e hijos se debe a la herencia de ciertas características y que los cambios que se observan de una generación a otra tienen como fin hacer a los nuevos organismos más aptos para sobrevivir. En la época de Darwin, estaba de moda una teoría llamada de la recombinación según la cual los padres heredaban a los hijos ciertas características que se mezclaban o recombinaban de alguna manera en ellos, pero no explicaba la aparición en nuevos individuos con características que no habían sido observadas en sus ancestros.

La teoría de la recombinación quedó desechada cuando el monje austriaco Gregor Mendel descubrió, como resultado de experimentos con chícharos, tres leyes básicas que gobiernan la herencia: la Ley de Segregación[1], la Ley de Independencia[2] y la Ley de la Uniformidad[3] [17].

Posterior al trabajo de Mendel, el botánico danés Hugo De Vries redescubrió las leyes de la herencia (Ley de Segregación, Ley de Independencia y Ley de la Uniformidad) pero además, motivado al descubrir una flor roja entre una gran cantidad de flores amarillas, desarolló la teoría de las mutaciones espontáneas que si bien no era del todo correcta ha servido para complementar la teoría evolutiva de Darwin. Según De Vries, los cambios en las especies no eran graduales sino más bien abruptos y, de hecho, aleatorios [2].

A continuación se hablará de estos trabajos de forma más detallada.

[1] Mendel planteó que los genes se encuentran agrupados en parejas en las células somáticas y que se segregan durante la formación de las células sexuales (gametos femeninos o masculinos). Cada miembro del par pasa a formar parte de células sexuales distintas. Cuando un gameto femenino y otro masculino se unen, se forma de nuevo una pareja de genes en la que el gen dominante oculta al gen recesivo

[2] En él se afirma que la expresión de un gen, para dar una característica física simple, no está influida, generalmente, por la expresión de otras características

[3] Esta última establece que las características heredadas de padres a hijos depende de si los genes de los padres son dominantes o recesivos.

Hoy en día, se conoce como Neo-Darwinismo [4] al paradigma resultante de unir las ideas de Darwin, Weismann y Mendel. La teoría Neo-Darwiniana establece que toda la diversidad de vida en el planeta puede ser explicada mediante los siguientes cuatro procesos:

- Reproducción, esta consiste en el apareamiento de dos individuos de la misma especie, dando como resultado nuevos individuo.
- Mutación, se da cuando un individuo de la población cambia algunas características de su fisonomía, lo cual lo hace diferente con los de su especie.
- Competencia, esta se aplica por la supervivencia de los mas fuertes. Dos individuos compiten para decidir quien de los dos sigue en la población.
- Selección, sirve para ver quienes son los mejores individuos de la población.

2.2.1. Seleccionismo de Weismann

August Weismann (1834-1914), formuló una teoría denominada del germoplasma, según la cual el cuerpo se divide en células germinales que pueden transmitir información hereditaria y en células somáticas, que no pueden hacerlo [18].

El francés Jean Baptiste Pierre Antoine de Monet (Caballero de Lamarck) [1] en su filosofía zoológica que data de 1809, expuso su teoría de

que las características adquiridas por un individuo durante su vida, son codificadas genéticamente y heredadas a sus descendientes de manera directa. Esto es, que el ambiente actúa directamente sobre las características físicas de los individuos, pues cuanto más se usa una estructura más se acentúa y viceversa.

Esto se conocería más tarde como la doctrina del uso y desuso. Esta teoría Lamarckiana fue refutada de manera implícita por primera vez, por el cientíco alemán August Weismann, al afirmar que iniciando con el óvulo fertilizado, existen dos procesos independientes de división de la célula. Uno conlleva al cuerpo o 'soma', (hoy conocido como fenotipo), y el otro (la línea germinal, hoy conocido como genotipo) conlleva a los gametos que forman el punto de inicio de la siguiente generación. De estas dos líneas celulares, el soma moriría, pero la línea germinal es potencialmente inmortal. La línea germinal es capaz de transmitir información hereditaria, mientras que el soma no puede hacerlo. La conclusión de Weismann es conocida como la teoría del germoplasma, y afirma que la selección natural es el único mecanismo capaz de cambiar la línea germinal, mientras que tanto esa línea germinal como el ambiente pueden influenciar al soma [3].

2.2.2. Genética de Gregor Mendel

El científico austriaco Gregor Johann Mendel[2] (1822-1884), fue pionero en el estudio sobre la transmisión de los caracteres hereditarios. En 1865 presentó su trabajo a la Sociedad de Historia Natural de Bron, con el título *Híbridos en plantas* [17]. En él se resumían experimentos que había llevado

a cabo durante 8 años usando chícharos (Pisum sativum) y explica cómo van a ser las características de los descendientes, a partir del conocimiento de las características de los progenitores.

Sus experimentos son el paradigma del análisis genético y su trabajo es considerado fundamental en la genética. Un diseño experimental sencillo junto con un análisis cuantitativo de sus datos fue la fuerza principal de su trabajo.

Los experimentos demostraron que:

- La herencia se transmite por elementos particulados (refutando, por tanto, la herencia de las mezclas) y que
- Siguen normas estadísticas sencillas.

Con la anterior conclusión Mendel desarrolló las siguientes 3 leyes:

Ley de la uniformidad Establece que si se cruzan dos razas puras para un determinado carácter, los descendientes de la primera generación son todos iguales entre sí e iguales a uno de los progenitores.

Ley de la disyunción Establece que los caracteres recesivos, al cruzar dos razas puras, quedan ocultos en la primera generación, reaparecen en la segunda en proporción de uno a tres respecto a los caracteres dominantes.

Ley de la segregación independiente Establece que los caracteres son independientes y se combinan al azar.

2.2.3. Teoría de la Evolución de Charles Darwin

La publicación del libro titulado *El Origen de las Especies* por el biólogo inglés Charles Darwin [4], en el año de 1859, vendría a revolucionar las ciencias biológicas e inclusive la manera de cómo el hombre se ve así mismo, ahora como resultado de un largo proceso de adaptación al medio ambiente.

A través de sus largos viajes, Darwin se dio cuenta de que la especie que no sufriera cambios se volvería incompatible con el entorno, puesto que éste tiende a cambiar con el tiempo. Asimismo, las similitudes entre padres e hijos observada en la naturaleza, le sugirieron que ciertas características de las especies eran hereditarias y de generación en generación ocurrían cambios cuya principal motivación era hacer a los nuevos individuos más aptos para sobrevivir.

De tal manera, Darwin infirió que debía haber una competencia por la sobrevivencia, existiendo especies con más capacidad de sobrevivir que otras. Darwin aceptó la teoría Lamarckiana bajo el término de *uso* y *desuso*, pero pensaba que la selección natural es lo más importante. Sus conclusiones ponen al descubierto la necesidad de tener una explicación a las variaciones en los seres vivos. La respuesta a ello sería encontrada años después por Gregor Mendel [4].

Tras su regreso a Inglaterra en 1836, Darwin comenzó a recopilar sus ideas acerca del cambio de las especies en sus Cuadernos sobre la transmutación de las especies. La explicación de la evolución de los organismos le surgió tras la lectura del libro *Ensayo sobre el principio de la población* (1798)

del economista británico Thomas Robert Malthus, que explicaba cómo se mantenía el equilibrio en las poblaciones humanas [4].

Malthus sostenía que ningún aumento en la disponibilidad de alimentos básicos para la supervivencia del ser humano podría compensar el ritmo de crecimiento de la población. Éste, por consiguiente, sólo podía verse frenado por limitaciones naturales, como las hambrunas o las enfermedades, o por acciones humanas como la guerra.

2.3. Neo-Darwinismo

La teoría evolutiva propuesta originalmente por Charles Darwin en combinación con el seleccionismo de August Weismann y la genética de Gregor Mendel, se conoce hoy en día como el paradigma Neo-Darwiniano [9].

El Neo-Darwinismo establece que la historia de la vasta mayoría de la vida en nuestro planeta puede ser explicada a través de un puñado de procesos estadísticos que actúan sobre y dentro de las poblaciones y especies: la reproducción, la mutación, la competencia y la selección [9].

La reproducción es una propiedad obvia de todas las formas de vida de nuestro planeta, pues de no contar con un mecanismo de este tipo, la vida misma no tendría forma de producirse.

En cualquier sistema que se reproduce a sí mismo continuamente y que está en constante equilibrio, la mutación está garantizada [21]. El contar con una cantidad finita de espacio para albergar la vida en la Tierra garantiza la

existencia de la competencia. La selección se vuelve la consecuencia natural del exceso de organismos que han llenado el espacio de recursos disponibles. La evolución es, por lo tanto, el resultado de estos procesos estocásticos (es decir, probabilísticos) fundamentales que interactúan entre sí en las poblaciones, generación tras generación.

2.4. Principales Paradigmas de Computación Evolutiva

Esta sección está dedicada a proporcionar algunos conceptos básicos de la computación evolutiva que se consideran importantes para establecer claramente el problema que deseamos solucionar. Nos enfocaremos en uno de sus paradigmas más conocidos que son los algoritmos genéticos.

2.4.1. Programación Evolutiva

Fue propuesta por Lawrence J. Fogel en 1960. En ella, el aprendizaje se ve como un comportamiento adaptativo [6]. Su interés está centrado principalmente en las relaciones entre padres e hijos, más que en el efecto de los operadores genéticos. La programación evolutiva es una abstracción de la evolución a nivel de las especies, por lo que no se aplica ningún operador de cruza, ya que individuos de diferentes especies no se pueden cruzar entre sí. Su operador único es pues la mutación, y la selección es probabilìstica.

El algoritmo básico de Programación Evolutiva se muestra en el **Algoritmo 2.1**.

Algorithm 2.1: PROGRAMACIONEVOLUTIVA(Tp, Pm, NG)

comment: Genera aleatoriamente una población inicial

for $i \leftarrow 0$ **to** Tp

  do $GeneraElemento(i)$

for $i \leftarrow 0$ **to** NG

comment: Aplica mutación

$OperadorMutacion()$

comment: Calcula aptitud de cada hijo

$OperadorAptitud()$

comment: selecciona por torneo

$OperadorSeleccionar()$

La propuesta original es la siguiente: una población de máquinas de estado finito (máquinas de Mealy [4]) se expone a una serie de símbolos de entrada (el ambiente) y se espera que, eventualmente, sea capaz de predecir las secuencias futuras de símbolos que recibirá. Fogel utilizó la función de pago que indicaba qué tan bueno era un cierto autómata para predecir un símbolo.

Para cada máquina padre, conforme se ofrece cada símbolo de entrada

[4] La máquina de Mealy es un transductor determinístico de estado finito que emite un símbolo de salida en cada movimiento. Para cada posible símbolo de entrada, en cada estado, la máquina puede generar un símbolo de salida y el siguiente estado de transición. A diferencia de las máquinas de Moore, cuya salida está determinada únicamente en función del estado presente, la salida de las máquinas de Mealy está en función del estado presente y la entrada [6].

a la máquina, se compara cada símbolo de salida con el siguiente símbolo de entrada. El valor (normalmente numérico) de esta predicción se mide posteriormente con respecto a la función de pago. Después de efectuarse la última predicción, una función de pago de cada símbolo indica la aptitud de la máquina.

2.4.2. Estrategias Evolutivas

Las Estrategias Evolutivas (EE) se desarrollaron en 1964 en Alemania con el objeto de resolver problemas de aerodinámica con alto grado de complejidad. Los investigadores relacionados fueron Ingo Rechenberg, Hans-Paul Schwefel y Paul Bienert [5].

Las EE trabajan con una abstracción a nivel de los individuos, por lo que, en este caso, sí existe un operador de cruza, sexual (dos padres) o panmítica (un solo padre). Sin embargo, la cruza es un operador secundario. La mutación es el operador primario. Estos valores son los que permiten que las EE sean auto-adaptativas (las estrategias evolutivas se evoluciona no sólo a las variables del problema, sino también a los parámetros mismos de la técnica). Su selección es determinística. Esta técnica opera a nivel fenotípico (es decir, no requieren codificación de las variables del problema).

La versión original, llamada (1 + 1)-EE no contempla el concepto de población. Existe un solo padre y a partir de él se genera un nuevo individuo mediante la siguiente expresión: $x_{t+1} = x_t + N(0; \bar{\sigma})$ donde t se refiere a

la generación y $N(0;\bar{\sigma})$ es un vector de números Gaussianos independientes con media cero y desviación estándar $\bar{\sigma}$. Si el hijo es mejor en aptitud, se mantiene; de lo contrario, se elimina.

La EE llamada $(\mu + 1)$-EE utiliza una población. Hay μ padres y se genera un solo hijo, el cual puede reemplazar al peor padre.
Por otro lado, Schwefel introdujo en 1975 el uso de múltiples hijos en las EE llamadas $(\mu + \lambda)$ EE y (μ, λ)-EE.

1. En la primera, los mejores μ mejores individuos obtenidos de la unión de padres e hijos se preservarán (población traslapable).

2. En la segunda, sólo los mejores μ hijos de la siguiente generación sobreviven (población no traslapable).

2.4.3. Algoritmos Genéticos

Los AG pueden verse como una familia de procedimientos de búsqueda adaptivos. Estos algoritmos fueron propuestos por John Holland a principio de los 1960s [9] motivado por resolver problemas de aprendizaje de máquina.

Su nombre se deriva del hecho de que están basados en modelos de cambio genético en una población de individuos. Esto es, se basan en:

- la noción Darwiniana de aptitud (fitness) que influye en generaciones futuras.

- el apareamiento que produce descendientes en generaciones futuras.

- un conjunto de operadores genéticos que determinan la configuración genética de los descendientes (tomada de los padres).

Un punto clave de estos modelos, es que el proceso de adaptación no se hace cambiando incrementalmente una sola estructura, sino manteniendo una población de estructuras a partir de las cuales se generan nuevas estructuras usando los operadores genéticos.

La estructura en la población está asociada con una aptitud y los valores se usan en competencia para determinar qué estructuras serán usadas para formar nuevas estructuras.

Una de sus características es su habilidad de explotar información acumulada acerca de un espacio de búsqueda inicialmente desconocido para guiar la búsqueda subsecuente a subespacios útiles.

Su aplicación está enfocada sobre todo a espacios de búsqueda grandes, complejos y poco entendidos.

Los pasos que sigue un AG son los siguientes:

Se generar una población inicial, normalmente de manera aleatoria. Los individuos de tal población serán un conjunto de cromosomas o cadenas que representen las posibles soluciones del problema, aunque en general no es posible representar todo el espacio de las soluciones (se requiere normalmente una codificación de las variables del problema).

Se aplicar una función de aptitud a cada uno de esos cromosomas, la cual se calcula de acuerdo a la solución codificada por el cromosoma.

Conociendo la aptitud de cada individuo, se lleva a cabo un proceso de selección (normalmente probabilística), que determina a los padres de la siguiente generación (presumiblemente los más aptos).

Se realiza un proceso de cruza, entre los padres que resultaron seleccionados en el paso anterior.

Se aplicar un operador de mutación a los hijos resultantes que serán ahora la nueva generación.

Este procedimiento se cicla desde el punto 2 hasta que se satisfaga algún criterio de terminación predefinido.

El algoritmo genético simple [4] es el que se muestra en el **Algoritmo 2.2**:

Algorithm 2.2: GenéticoSimple(Tp, Pc, Pm, NG)

comment: Generar una población inicial

for $i \leftarrow 1$ **to** Tp

  do $GeneraElemento()$

for $p \leftarrow 1$ **to** NG

  comment: Aplicar una función de aptitud

  for $i \leftarrow 1$ **to** Tp

    do $FuncionAptitud()$

  comment: Aplicar una función de Seleccion

  for $i \leftarrow 1$ **to** Tp

    do $FuncionSeleccion()$

  comment: Aplicar una función de Cruza

  for $i \leftarrow 1$ **to** Tp

    do $FuncionCruza(Pc)$

  comment: Aplicar una función de Mutación

  for $i \leftarrow 1$ **to** Tp

    do $FuncionMutacion(Pm)$

Resumen

La computación evolutiva, se refiere a un conjunto de técnicas de búsqueda y optimización inspiradas en el proceso de selección natural. La computación evolutiva es una de las usadas para resolver problemas no lineales de optimización.

Los antecedentes en los que se basa esta teoría son: el seleccionismo de Weismann, la genética de Gregor Mendel y el principio de selección natural de Charles Darwin.

El seleccionismo de Weismann habla sobre la teoría del germen plasmático, concepto según el cual una sustancia especial constituye la única continuidad orgánica entre una generación y la siguiente.

La genética de Gregor Mendel, explica todo sobre la transmisión de los caracteres hereditarios. Esto se derivó de experimentos que Mendel llevó a cabo durante 8 años usando chícharos y explica cómo van a ser las características de los descendientes, a partir del conocimiento de las características de sus progenitores.

La selección natural de Charles Darwin, infirió en que debía haber una competencia por la sobrevivencia, existiendo especies con más capacidad de sobrevivir que otras, lo cual explica las variaciones existentes en los seres vivos.

Los principales paradigmas de la computación evolutiva son: Programación Evolutiva, Estrategias Evolutivas y Algoritmos Genéticos.

La Programación Evolutiva es una abstracción de la evolución a nivel de las especies, por lo que no se aplica ningún operador de cruza, ya que

individuos de diferentes especies no se pueden cruzar entre sí. Su operador único es la mutación, y la selección es probabilística.

Las Estrategias Evolutivas constituyen con una abstracción a nivel de los individuos, por lo que sí existe un operador de cruza, la cual puede ser sexual (dos padres) o panmítica (un solo padre). Sin embargo, la cruza aquí es un operador secundario. La mutación es el operador primario. Su selección es determinística.

Los AG pueden verse como una familia de procedimientos de búsqueda adaptivos. Un punto clave de estos modelos, es que el proceso de adaptación no se hace cambiando incrementalmente una sola estructura, sino manteniendo una población de estructuras a partir de las cuales se generan nuevas estructuras usando los operadores genéticos. La estructura en la población está asociada con una aptitud y los valores se usan en competencia para determinar qué estructuras serán usadas para formar nuevas estructuras.

En el capítulo siguiente se describe de forma mas detallada lo que son los AG, explicándose a detalle sus mecanismos tales como los tipos de selección que se pueden utilizar dentro de un AG, así como los tipos de cruza y mutación que se pueden incluir en este tipo de algoritmos.

Capítulo III

Algoritmos Genéticos

3. Algoritmos Genéticos

En este capítulo estudiaremos la base teórica que sustenta a los algoritmos genéticos y que necesitaremos para nuestro problema. Justificaremos por qué hemos escogido los algoritmos genéticos para optimizar las exponenciaciones Modulares.

3.1. Introducción

John Holland [7] desde pequeño, se preguntaba cómo logra la naturaleza crear seres cada vez más perfectos (aunque, como se ha visto, esto no es totalmente cierto, o en todo caso depende de qué entienda uno por perfecto). Lo curioso era que todo se llevaba a cabo a base de interacciones locales entre individuos, y entre éstos y lo que les rodea.

No sabía la respuesta, pero tenía una cierta idea de cómo hallarla: tratando de hacer pequeños modelos de la naturaleza, que tuvieran alguna de sus características, y ver cómo funcionaban, para luego extrapolar sus conclusiones a la totalidad. De hecho, ya de pequeño hacía simulaciones de batallas célebres con todos sus elementos: copiaba mapas y los cubría luego de pequeños ejércitos que se enfrentaban entre sí.

En los años 50 entró en contacto con las primeras computadoras, donde pudo llevar a cabo algunas de sus ideas, aunque no se encontró con un ambiente intelectual fértil para propagarlas. Fue a principios de los 60, en la Universidad de Michigan en Ann Arbor, donde, dentro del grupo Logic of Computers, sus ideas comenzaron a desarrollarse y a dar frutos. Y fue, además, leyendo un libro escrito por un biólogo evolucionista, R. A. Fisher,

titulado *La teoría genética de la selección natural*, como comenzó a descubrir los medios de llevar a cabo sus propósitos de comprensión de la naturaleza. De ese libro aprendió que la evolución era una forma de adaptación más potente que el simple aprendizaje, y tomó la decisión de aplicar estas ideas para desarrollar programas bien adaptados para un fin determinado [7].

3.2. Características Generales de los AG

- Son algoritmos estocásticos. Dos ejecuciones distintas pueden dar dos soluciones distintas.
- Son algoritmos de búsqueda múltiple, por lo que producen varias soluciones en una sola ejecución. Aunque habitualmente las aptitudes de los individuos de la población final es similar, los individuos suelen ser distintos entre sí.
- De todos los algoritmos de optimización estocásticos, los algoritmos genéticos se encuentran entre los más exploratorios disponibles, por lo que suelen producir soluciones muy buenas a problemas complejos.
- En los algoritmos genéticos (salvo poblaciones iniciales realmente degeneradas, en las que el operador de mutación va a tener mucho trabajo) la convergencia del algoritmo es poco sensible a la población inicial si ésta se escoge de forma aleatoria (siguiendo una distribución uniforme) y es lo suficientemente grande.
- La curva de convergencia asociada suele ser muy rápida al principio, al grado de que casi enseguida se bloquea como se muestra en la figura siguiente. Esto se debe a que el algoritmo genético es excelente

para descartar subespacios realmente malos. Cada cierto tiempo, la población vuelve dar el salto evolutivo, y se produce un incremento en la velocidad de convergencia excepcional. La curva de convergencia se puede observar en la figura 1.

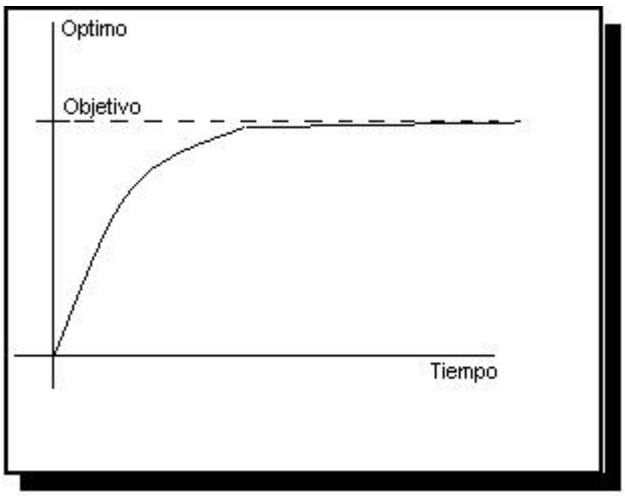

Figura 1: Curva de convergencia de un AG

- Todo el proceso del AG depende de la representación adoptada para codificar las soluciones. Este es el concepto clave dentro de los algoritmos genéticos, ya que una buena codificación puede hacer la programación y la resolución muy sencillas o muy difíciles [8].

Algoritmo Básico

El AG enfatiza la importancia de la cruza sexual (operador principal) sobre el de la mutación (operador secundario), y usa selección probabilística. El AG básico es el que se muestra en el **Algoritmo 3.1** :

Algorithm 3.1: GenéticoBásico(Tp, Pc, Pm, NG)

```
for i ← 1 to Tp
  do GeneraElemento()
for p ← 1 to NG
{ for i ← 1 to Tp
    do FuncionAptitud()
  for i ← 1 to Tp
    do FuncionSeleccion()
  for i ← 1 to Tp
    do FuncionCruza()
  for i ← 1 to Tp
    do FuncionMutacion()
```

3.3. Tamaño de la Población Inicial

Goldberg [9] realizó un estudio teórico del tamaño ideal de la población de un AG en función del número esperado de nuevos esquemas por miembro de la población. Usando una población inicial generada aleatoriamente con igual probabilidad para el cero y el uno, Goldberg derivó la siguiente expresión: Tam Población = 1,65($2^{(0,21L)}$) donde: L = longitud de la cadena (binaria). Esta expresión sugiere tamaños de población demasiado grandes para cadenas de longitud moderada. Considermos los siguientes ejemplos:

L = 30 Tam Población = 130

L = 40 Tam Población = 557

L = 50 Tam Población = 2389

L = 60 Tam Población = 10244

La Figura 2 refleja los resultados anteriores.

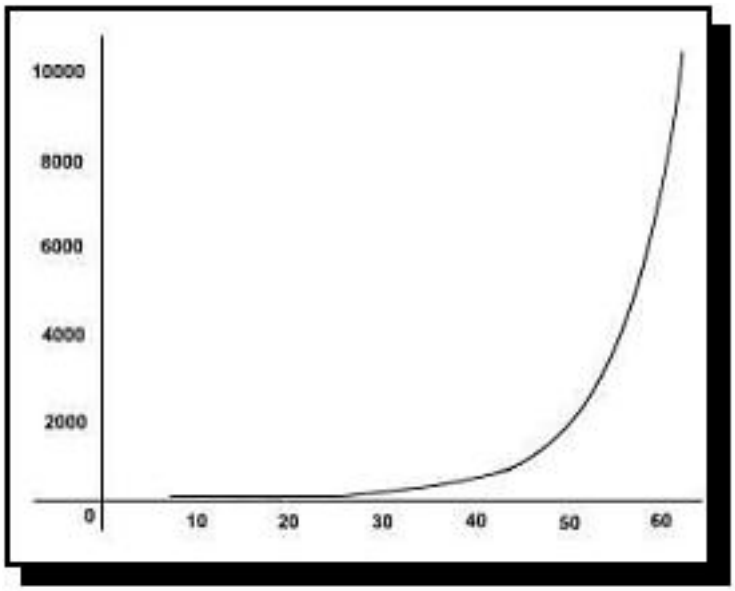

Figura 2: Tamaño optimo de población para un algoritmo genético según Goldberg

Según el teorema de los esquemas, un AG procesa $O(N^3)$ esquemas. A partir de esta idea, Goldberg concluye entonces que a mayor valor de N (tamaño de la población), mejor desempeño tendrá el AG, y de ahí deriva su expresión para calcular el tamaño óptimo de una población. Sin embargo, posteriormente se pudo demostrar que la expresión derivada por Goldberg

estaba equivocada.

La regla empírica más común es usar una población de al menos 2 veces L. Esta regla sólo se menciona para que el lector la tome en cuenta, pero no es usada en este trabajo.

3.4. Valor de Aptitud de un Individuo

Durante la evaluación, se decodifica el gen, convirtiéndose en una serie de parámetros de un problema, se halla la solución del problema a partir de esos parámetros, y se le da una puntuación a esa solución en función de lo cerca que esté de la mejor solución. A esta puntuación se le llama aptitud.

3.5. Selección

Para aplicar los operadores genéticos tendremos que seleccionar un subconjunto de la población.

Los individuos que tengan algún rasgo que los haga menos válidos para realizar su tarea de seres vivos, no llegarán a reproducirse, y, por tanto, su patrimonio genético desaparecerá de la población; algunos no llegarán ni siquiera a nacer.

Algunas de las técnicas de selección que disponemos son descritas en los siguientes apartados de esta sección.

3.5.1. Selección Proporcional

Este nombre describe a un grupo de esquemas de selección originalmente propuestos por Holland en los cuales se eligen individuos de acuerdo a su

contribución de aptitud con respecto al total de la población. Se suelen considerar 4 grandes grupos dentro de las técnicas de selección proporcional [9]:

1. Selección mediante Ruleta
2. Selección de Sobrante Estocástico
3. Selección de Universal Estocástica
4. Selección de Muestreo Determinístico

En este documento sólo describirá el método de La Ruleta. Para mayor información sobre los métodos restantes se sugiere consultar la referencia [9]

Selección mediante Ruleta

Esta técnica fue propuesta por DeJong [9], y ha sido el método proporcional más comúnmente usado desde los orígenes de los algoritmos genéticos. Cada inidividuo de la población tiene asignado un rango proporcional o inversamente proporcional a su adaptación.

El número al azar será un número aleatorio forzosamente menor que el tamaño del rango. El elemento escogido será aquel en cuyo rango esté el número resultante de sumar el número aleatorio con el resultado total que sirvió para escoger el elemento anterior.

A continuación se describe el algoritmo de selección por Ruleta [9], en el

Algoritmo 3.2

- Calcular la suma de valores esperados T

- Repetir N veces (N es el tamaño de la población):

  1. Generar un número aleatorio **r** entre 0.0 y T.
  2. Ciclar a través de los individuos de la población sumando los valores esperados hasta que la suma sea mayor o igual a **r**.
  3. El individuo que haga que esta suma exceda el límite es el seleccionado.

Algorithm 3.2: SELECCIÓNPORRULETA(N)

comment: Calcular la suma de valores esperados T

$Suma \leftarrow 0$

for $i \leftarrow 1$ **to** N

  do $Suma = Aptitud[i] + Suma$

for $i \leftarrow 1$ **to** N

  do $ValorEsperado[i] = Aptitud[i]/Suma$

$T \leftarrow 0$

for $i \leftarrow 1$ **to** N

  do $T = ValorEsperado[i] + T$

for $i \leftarrow 1$ **to** N

- **comment:** Generar un número aleatorio **r** entre 0.0 y T
- $r \leftarrow GENERAR(0 \cdot 0, T)$
- $Suma \leftarrow 0$
- **for** $i \leftarrow 1$ **to** N
  - $Suma = ValorEsperado[i] + Suma$
  - **if** $Suma \geq r$
    - **comment:** Selecciona al individuo i
    - $SeleccionaIndividuo(i)$

3.5.2. Selección Mediante Torneo

Esta técnica fue propuesta por Wetzel [19] y fue estudiada en la tesis doctoral de Brindle [20].

Este operador escoge un subconjunto de individuos habitualmente de forma aleatoria o estocástica y, de entre ellos, selecciona el más adecuado. Esta técnica tiene la ventaja de que permite un cierto grado de elitismo (el mejor nunca va a morir, y los mejores tienen más probabilidad de reproducirse y de emigrar que los peores) pero sin producir una convergencia genética prematura, esto es porque el que tiene la mejor aptitud nunca va a perder al ser comparado con otro individuo que tenga menos aptitud. Por ejemplo.

Individuo	Aptitud
Ind 1	10
Ind 2	7

Individuo seleccionado: **Ind 1**

La idea básica del método es seleccionar con base en comparaciones directas de los individuos al mejor.

Hay 2 versiones de la selección mediante torneo:

- Determinística
- Probabilística

El algoritmo de **Selección por Torneo versión determinística** se muestra en **Algoritmo 3.3**.

Algorithm 3.3: SELECTORNEODETERMINISTICA($Tp, NumInd$)

for $i \leftarrow 1$ **to** Tp

comment: Escoger un número p de individuos (típicamente 2).

$EscogerIndividuos(NumInd)$

comment: Compararlos con base en su aptitud.

$CompararIndividuos()$

comment: El ganador del torneo es el individuo más apto

$SeleccionarGanador()$

El algoritmo de la **versión probabilística** es idéntico al anterior, excepto por el paso en que se escoge al ganador. En vez de seleccionar siempre al individuo con aptitud más alta, se aplica *flip(p)* y si el resultado es cierto, se selecciona al más apto. De lo contrario, se selecciona al menos apto.

El valor de p permanece fijo a lo largo de todo el proceso evolutivo y se escoge en el siguiente rango:

$$0{,}5 < p \leq 1$$

Observe que si $p = 1$, la técnica se reduce a la versión determinística.

3.5.3. Selección de Estado Uniforme

Esta técnica fue propuesta por Whitley [9] y se usa en Algoritmos Genéticos no generacionales, en los cuales sólo unos cuantos individuos son reemplazados en cada generación (los menos aptos). Esta técnica suele usarse

cuando se evolucionan sistemas basados en reglas (p.ej., sistemas de clasificadores) en los que el aprendizaje es incremental.

En general, la técnica resulta útil cuando los miembros de la población resuelven colectivamente (y no de manera individual) un problema.

El algoritmo de la selección de estado uniforme se puede ver en el **Algoritmo 3.4**.

Algorithm 3.4: SELECESTADOUNIFORME(Tp)

$G \leftarrow PoblacionOriginal()$

comment: Seleccionar R individuos ($1 \leq R < M$)

$H \leftarrow SeleccionaIndividuo(R)$

for $i \leftarrow 1$ **to** R

- **comment:** Aplicar operador de cruza
- $OperadorCruza()$
- **comment:** Aplicar operador de mutación
- $OperadorMutacion()$

comment: Elegir a los mejores individuos en H.

comment: Reemplazar peores individuos de G por mejores de H.

3.6. Cruza

Cuando las dos células sexuales, o gametos, una masculina y otra femenina se combinan, los cromosomas de cada una también lo hacen, intercambiándose genes, que a partir de ese momento pertenecerán a un cromosoma diferente. A veces también se produce traslocación dentro de un cromosoma; una secuencia de código se elimina de un sitio y aparece en otro sitio del cromosoma, o en otro cromosoma.

En computación evolutiva se simula la cruza intercambiando segmentos de cadenas lineales de longitud fija (los cromosomas).

La cruza consiste en el intercambio de material genético entre dos cromosomas. La cruza es el principal operador genético, hasta el punto que se puede decir que no es un algoritmo genético si no tiene cruza, y, sin embargo, puede serlo perfectamente sin mutación, según descubrió Holland [7].

Para aplicar la cruza, se escogen aleatoriamente dos miembros de la población. Si se cruzan dos individuos iguales, la cruza no tiene efecto, por lo cual, la población tenderá a converger a una solución única, conforme se pierda diversidad (o sea, todos los individuos sean iguales, o muy parecidos).

3.6.1. Cruza de Un Punto

Esta técnica fue propuesta por Holland [9], y fue muy popular durante muchos años.

Se selecciona un punto al azar de la cadena.

Hijo 1: La parte previa al punto de cruza es copiada del genoma del padre 1 y pegada al hijo 1. La parte posterior del hijo 1 se copia de la parte posterior del padre 2 .

Hijo 2: La parte previa al punto de cruza es copiada del genoma del padre 2 y pegada al hijo 2. La parte posterior del hijo 2 se copia de la parte posterior del padre 1.

La cruza de un punto se ilustra en la figura 3.

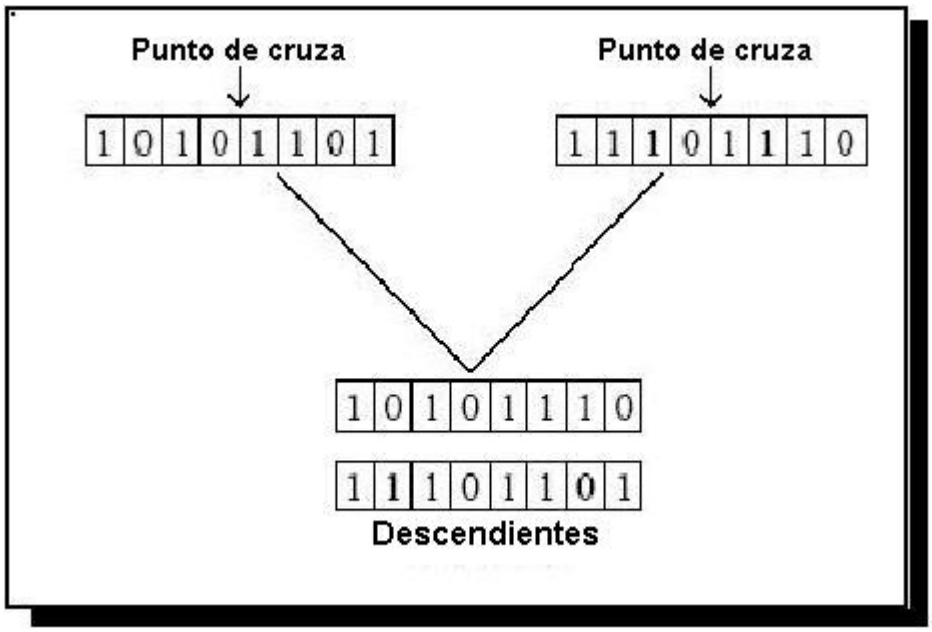

Figura 3: Representación de la cruza de un punto

3.6.2. Cruza de Dos Puntos

DeJong fue el primero en implementar una cruza de n puntos, como una generalización de la cruza de un punto. El valor $n = 2$ es el que minimiza los efectos disruptivos (o destructivos) de la cruza y de ahí que se a usado con gran frecuencia [10].

En la cruza de dos puntos, se eligen dos posiciones al azar, en la longitud de la cadena. El hijo 1 se forma tomando la posición previa al primer punto de cruza y la posterior al segundo punto de cruza del padre 1. La parte de enmedio se toma del padre 2. El hijo 2 se construye de manera análoga.

En general, sin embargo, es aceptado que la cruza de dos puntos es mejor que la cruza de un punto. Asimismo, el incrementar el valor de n se asocia con un mayor efecto disruptivo de la cruza [9].

La cruza en dos puntos se puede observar en la Figura 4.

3.6.3. Cruza Uniforme

Esta técnica fue propuesta originalmente por Ackley, aunque se le suele atribuir a Syswerda [9].

Se genera un patrón aleatorio de 1s y 0s, y se intercambian los bits de los dos cromosomas que coincidan donde hay un 1 en el patrón. Otra alternativa es generar un número aleatorio para cada bit, y si supera una determinada probabilidad se intercambia ese bit entre los dos cromosomas. La cruza uniforme se puede ver en la figura 5.

La cruza uniforme tiene un mayor efecto disruptivo que cualquiera de

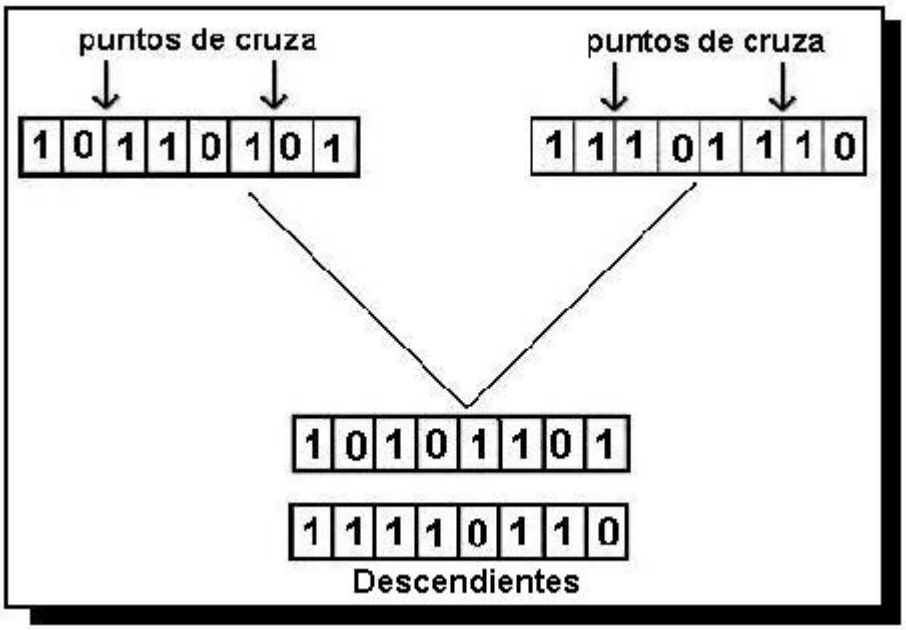

Figura 4: Representación de la cruza en dos puntos

las 2 cruzas anteriores. A fin de evitar un efecto excesivamente disruptivo, suele usarse con $Pc = 0{,}5$ [9]

3.7. Mutación

En la evolución natural, una mutación es un suceso muy poco común. Un ejemplo de mutación en la naturaleza sería una planta que está siendo atacada por una plaga de gusanos. La planta tiene la obligación de protegerse si es que no quiere ser exterminada y desaparecer del planeta. Entonces la planta tiende a mutar. Un ejemplo de mutación sería segregar algún veneno, crecer con espinas, etc., lo cual ayudaría a que sobreviva.

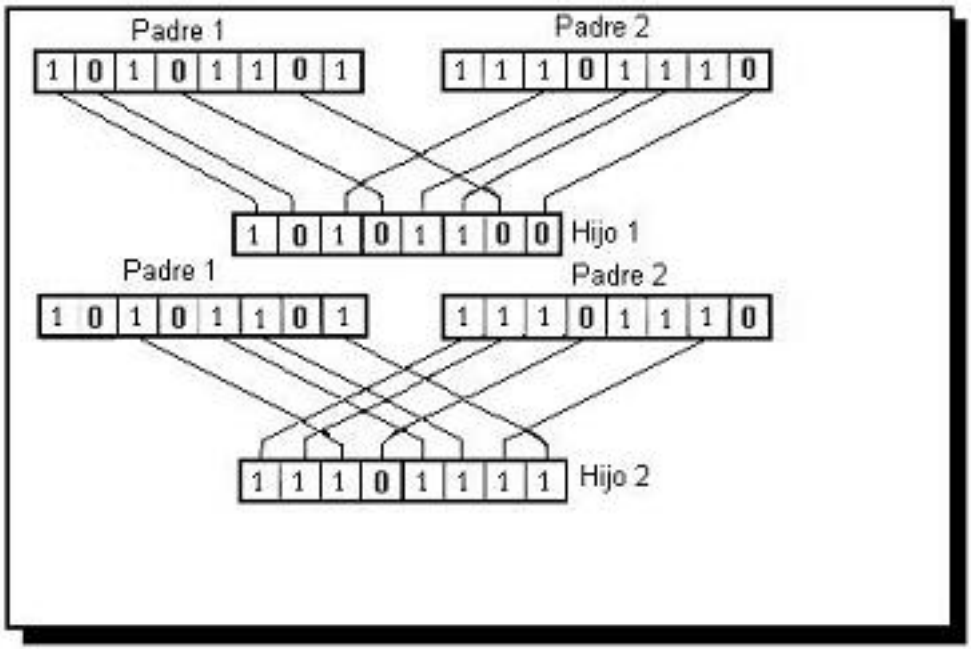

Figura 5: Representación de la cruza Uniforme

La mutación en los AG es algo similar. Algunos individuos tienden a mutar para elevar su aptitud y de esa manera poder seguir en la población.

En la mayoría de los casos las mutaciones son letales, pero en promedio, contribuyen a la diversidad genética de la especie. En un algoritmo genético tendrán el mismo papel, y la misma frecuencia (es decir, muy baja) [8].

La mutación se considera como un operador secundario en los algoritmos genéticos canónicos. Es decir, su uso es menos frecuente que el de la cruza. En la práctica, se suelen recomendar porcentajes de mutación de entre 0.001 y 0.01 para la representación binaria [9].

Una vez establecida la frecuencia de mutación, por ejemplo, uno por mil, se examina cada bit de cada cadena cuando se vaya a crear la nueva criatura a partir de sus padres (normalmente se hace de forma simultánea a la cruza). Si un número generado aleatoriamente está por debajo de esa probabilidad, se cambiará el bit (es decir, de 0 a 1 o de 1 a 0). Si no, se dejará como está. Dependiendo del número de individuos que haya y del número de bits por individuo, puede resultar que las mutaciones sean extremadamente raras en una sola generación.

La mutación es un mecanismo generador de diversidad, y, por tanto, es de gran ayuda para un AG está estancado. Sin embargo, su uso excesivo puede evitar, al hacer la búsqueda más aleatoria. Por ello es más conveniente usar otros mecanismos de generación de diversidad, tales como aumentar el tamaño de la población, en vez de elevar excesivamente el porcentaje de mutación [7].

Resumen

Los AG son algoritmos de optimización estocásticos y de búsqueda poblacional (o múltiple). Su convergencia suele ser también rápida, dado que desecha pronto a un gran conjunto de malas soluciones. El desempeño de un AG depende mucho de la representación. El AG enfatiza la importancia de la cruza (operador primario) sobre el de mutación (operador secundario), y usa selección probabilística.

En un AG se resaltan varios operadores importantes que afectan su desempeño: tamaño de la población, función de aptitud, selección, cruza y mutación.

La población inicial de un AG debe ser generada eleatoriamente. Varios autores concluyen que entre mayor sea el tamaño de población, el AG, tendrá un mejor desempeño.

La aptitud de cada individuo es para saber quién tiene la mejor solución, de tal manera que entre mejor sea la aptitud de un individuo, más probabilidades tiene de seguir o de tener descendintes en la siguiente generación.

El operador de selección se encarga de reunir a los individuos más valiosos de la generación los cuales realizan mejor su tarea de seres vivos. Los individuos que sean seleccionados son los que realizarán la tarea de cruza y mutación.

La cruza es el operador principal de los AG ya que si no hay cruza entonces no es un AG. La cruza se realiza siempre con dos individuos de la población, los cuales intercambian genes dando como resultado dos hijos, los cuales pasarán a formar la nueva población. Hay distintos tipos de cruza, de entre las cuales las más importantes son: cruza de un punto, cruza de dos puntos y cruza uniforme.

La mutación es un operador secundario en los AG. En la naturaleza la mutación es un suceso poco común. En los AG, es similar, ya que este operador se aplica con una probabilidad muy baja. La mutación es un mecanismo generador de diversidad, pero también es disruptivo, y su uso excesivo puede evitar la convergencia.

En el siguiente capítulo se explica el problema que se aborda en este trabajo de tesis. El cual consiste en encontrar el menor número de operaciones al elevar cierta cantidad a un exponente (α^e). Éste es un problema difícil de resolver de forma determista, ya que su complejidad es factorial. De ahí que se justifique el uso de AG para abordar este problema.

Capítulo IV

Exponenciaciones Modulares

4. Exponenciaciones Modulares

Las exponenciaciones modulares son simplemente la búsqueda de el menor número de operaciones para calcular una operación con exponentes. Por ejemplo si queremos saber cuanto es 3^9 basta con multiplicar $3 \times 3 \times 3 \times 3 \times 3 \times 3 \times 3 \times 3 \times 3$. Con este mecanismo se tienen que realizar ocho operaciones. En cambio si utilizamos las exponenciaciones modulares para este mismo problema (3^9) bastaría con saber la cadena de adición para el 9. Una cadena sería la siguiente:

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 9$$

Entonces, para calcular el valor tendría que hacer lo siguiente, ponemos el 3 luego multiplicamos $3 \times 3 = 3^2$ utilizando este resultado y el 3 podemos hacer lo siguiente $3^2 \times 3 = 3^3$. Para hacer un cálculo mas rápido tomamos el resultado anterior y lo multiplicamos consigo mismo, lo que nos da $3^3 \times 3^3 = 3^6$. Nótese que las potencias que se están encontrando son los números de la cadena de adición. Por tanto, lo que queda por hacer es $3^6 \times 3^3 = 3^9$. De esta manera logramos realizar la operación en tan solo cuatro operaciones y no en ocho como se hizo anteriormente, lo que muestra la potencia de esta técnica.

4.1. Introducción

Aún los sistemas más primitivos de llaves públicas y criptográficos requieren del cálculo de exponenciación modular como su bloque constructor principal. Por ello, el problema de desarrollar exponenciaciones modulares eficazmente ha atraído una atención considerable desde hace muchos años.

Se sabe que las cadenas óptimas de adición (o sea, las más pequeñas) son el principal concepto matemático para lograr exponenciaciones modulares óptimamente. Sin embargo, encontrar una cadena óptima de adición de r de largo es un problema NP-duro en el cual el tamaño del espacio de búsqueda es $r!$.

En esta tesis se explora el uso de un AG para el problema de descubrimiento de la cadena de adición óptima (la más pequeña). Se explicará el uso del AG en detalle y se analizan los resultados experimentales obtenidos, los cuales sugieren que los algoritmos evolutivos pueden ser una alternativa viable para solucionar este famoso problema en una forma casi óptima.

4.2. Importancia del Problema

Las exponenciaciones modulares son de gran importancia en sistemas de encriptación y en llaves públicas [14].

Además de su relevancia histórica y teórica, la exponenciación modular tiene muchas aplicaciones prácticas importantes en las áreas de códigos de corrección de errores y la criptografía. La exponenciación modular se usa en varios sistemas criptográficos cruciales, en llaves públicas como RSA y en el Agente De Sistema De Directorios (DSA). [16]

Por ejemplo, el esquema criptográfico RSA se basa en el cálculo de M^e mód n, donde e es un número fijo, M es un mensaje numérico arbitrariamente seleccionado y n es el producto de dos números primos grandes,

a saber, $n = pq$. Además, la exponenciación modular es también un bloque constructivo principal para varios problemas de teoría de números incluyendo números primos, factorización de números enteros, cálculo inverso multiplicativo, etc.

4.3. Descripción del problema

El problema de determinar la secuencia correcta de multiplicaciones requeridas para representar una Exponenciación Modular puede ser elegantemente formulada usando los conceptos de **Cadenas de adición** [16].

Sea α un entero arbitrario en el rango $[1, n-1]$, y e un número entero positivo arbitrario. Definimos a la exponenciación modular como el problema de encontrar el entero único $\beta \in [1, n-1]$ que satisface la ecuación

$$\beta = \alpha^e \text{ mód } n \tag{1}$$

En general, existen dos estrategias principales para calcular la ecuación (1) eficazmente. Un enfoque consiste en implementar la multiplicación modular, que es el bloque principal requerido para la exponenciación modular. El otro enfoque consiste en reducir el número total de multiplicaciones necesitadas para calcular β. Esta tesis describirá algunas de las técnicas más más recientes que se conocen para resolver este problema.

Formalmente, una cadena de adición e con longitud l es una secuencia U de enteros positivos, $u_0 = 1, u_1 \cdots, u_l = e$ tal que para cada $i > 1$, $u_i = u_j + u_k$ para algunos j y k con $0 \leq j \leq k < i$. Por consiguiente, si U

es una cadena de adición que calcula e, entonces para algún $\alpha \in [1, n-1]$ podremos encontrar $\beta = \alpha^e$ mód n calculando $\alpha, \alpha^{u_1}, \cdots, \alpha^{u_{l-1}}, \alpha^e$.

Sea $l(e)$ la longitud más corta de cualquier cadena válida de adición para un entero e positivo dado. Entonces, el número mínimo teórico de multiplicaciones requeridas para calcular la exponenciación modular (1) es precisamente $l(e)$.

Desafortunadamente, el problema de determinar una cadena de adición para e con la longitud más corta $l(e)$ es un problema **NP**-duro. Por eso tenemos que usar alguna estrategia heurística para encontrar una cadena de adición óptima al lidiar con exponentes e grandes .

El problema del que nos ocupamos en esta tesis consiste en encontrar la cadena más pequeña de adición para un exponente e. Formalmente, una cadena de adición puede ser definida como sigue.

Definición una *cadena de adición* U para un entero positivo e de longitud l es una secuencia de enteros positivos $U = \{u_0, u_1, \cdots, u_l\}$, y una secuencia asociada r de pares $V = \{v_1, v_2 \cdots, v_l\}$ con $v_i = (i_1, i_2)$, $0 \leq i_2 \leq i_1 < i$, tal que:

- $u_0 = 1$ y $u_l = e$;
- para cada $u_i, 1 \leq i \leq l, u_i = u_{i_1} + u_{i_2}$.

El espacio de búsqueda para calcular la cadena de adición óptima incrementa su tamaño en una proporción factorial existiendo $r!$ cadenas de adición diferentes y válidas con r de largo. Claramente, el problema de en-

contrar las más pequeñas se complica cada vez más conforme aumenta el valor del exponente.

Resumen

La exponenciación modular tiene muchas aplicaciones prácticas importantes en las áreas de códigos de corrección de errores y en criptografía.

El problema de determinar la secuencia correcta de multiplicaciones requeridas para representar una Exponenciación Modular puede ser elegantemente formulada usando los conceptos de cadenas de adición.

Una cadena de adición e con longitud l es una secuencia U de enteros positivos, $u_0 = 1, u_1 \cdots, u_l = e$ tal que para cada $i > 1$, $u_i = u_j + u_k$ para algunos j y k con $0 \leq j \leq k < i$. Por consiguiente, si U es una cadena de adición que calcula e, entonces para algún $\alpha \in [1, n-1]$ podremos encontrar $\beta = \alpha^e$ mód n calculando $\alpha, \alpha^{u_1}, \cdots, \alpha^{u_{l-1}}, \alpha^e$.

El espacio de búsqueda para calcular la cadena de adición óptima incrementa su tamaño en una proporción factorial existiendo $r!$ cadenas de adición diferentes y válidas con r de longitud.

En el próximo capítulo se explica ampliamente la forma en que se soluciona el problema de las exponenciaciones modulares usando un AG con representación entera. Además, se dará a conocer el pseudocódigo de cada uno de los operadores que se propusieron para el AG que se diseño para resolver el problema de las exponenciaciones modulares antes indicado.

Capítulo V

Algoritmo Propuesto

5. Algoritmo Propuesto

En este capítulo se ataca el problema de exponenciaciones modulares usando como herramienta un AG con representación entera. Aquí se describen los operadores de cruza y mutación para este problema. Además, se explica cómo generar la población y cómo se le puede asignar un valor de aptitud a un individuo. Las exponenciaciones modulares son un ejemplo concreto donde se pueden aplicar los AG y enseguida se explican los tipos de operadores que se usaron en este trabajo.

5.1. Introducción

En este trabajo, se presenta un AG que realiza una aproximación satisfactoria para encontrar cadenas de adición óptimas. Como indicamos antes, para tener éxito al aplicar un AG requerimos de una buena representación, una función de aptitud apropiada y un buen conjunto de operadores de cruza y mutación.

5.2. Tipo de Representación Usada

Una de las decisiones más difíciles que deben ser tomadas en cuenta cuando se diseña un AG es en torno al tipo más apropiado de representación para codificar las soluciones potenciales del problema de interés.

En este trabajo, se opta por una codificación entera , usando cromosomas de longitud variable. Cada elemento de la cadena de adición es directamente mapeada en cada gene en el cromosoma. En este caso, el genotipo y el

fenotipo son ambos lo mismo.

Por ejemplo, si minimizamos la cadena de adición para el exponente $e = 6271$, una solución puede ser

$$1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 10 \rightarrow 20 \rightarrow 30 \rightarrow 60 \rightarrow 90 \rightarrow 180 \rightarrow 360 \rightarrow 720 \rightarrow 1440 \rightarrow 2880 \rightarrow 5760 \rightarrow 5970 \rightarrow 6150 \rightarrow 6240 \rightarrow 6270 \rightarrow 6271$$

5.3. Generación de la Población Inicial

Al iniciar la ejecución del AG, se requiere de una primera población, la cual se genera aleatoriamente (de acuerdo con una distribución uniforme).

Cada cadena de adición debe de cumplir con la definición vista anteriormente. El **Algoritmo 5.1** describe el procedimiento para generar la población inicial.

Algorithm 5.1: PoblaciónInicial(N)

```
for i ← 0 to 2
{ for j ← 0 to N
  { Poblacion[i][j] ← i + 1
for j ← 0 to N
{ for i ← 0 to N
  { comment: NUM: definición de cadenas de adición
    Poblacion[j][i] ← NUM;
    if Poblacion[i][j] ← Objetivo
    { Aptitud[j] ← i + 1
      break
```

5.4. Selección Usada

El tipo se selección que se usa en este documento es Selección por Torneo. Esta técnica nos dice que se deben seleccionar p individuos. En nuestro caso, $p = 2$, esto quiere decir que el número de participantes en este operador tendrá que ser de dos. El ganador de cada torneo será el que tenga la mejor aptitud, v.g., el que tenga la cadena de adicción más corta. Por ejemplo si hay dos individuos que tienen las siguientes cadenas de adición:

Objetivo $e = 20$

cad 1: $1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 20$

cad 2: $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 18 \rightarrow 20$

como estamos tratando con un problema de minimización, entonces el ganador de este torneo sería **cad 1** por tener la longitud más corta.

El **Algoritmo 5.2** describe al operador de selección:

Algorithm 5.2: SELECCIÓNUSADA( )

for $i = 1$ **to** Tp
- $Ind1 \leftarrow GeneraAleatorio(1, Tp)$
- $Ind2 \leftarrow GeneraAleatorio(1, Tp)$
- **if** $Aptitud[Ind1] < Aptitud[Ind2]$
  - **then** $Padres[i] = Poblacion[Ind1]$
  - **else if** $Aptitud[Ind1] > Aptitud[Ind2]$
  - **then** $Padres[i] = Poblacion[Ind2]$
  - **else if** $Aptitud[Ind1] = Aptitud[Ind2]$
  - **then** $Padres[i] = Poblacion[Ind1]$

Como se puede observar en el pseudocódigo anterior, se van a seleccionar los N mejores padres. En esta selección se repetirán muchos individuos, porque la selección de competidores es de forma aleatoria y puede que un individuo participe varias veces. Todos los ganadores de este torneo pasarán a formar la población de padres. A esta nueva población se le aplicará el operador de cruza, que se explica a continuación.

5.5. Tipo de Cruza Utilizada

El operador de cruza crea a dos hijos de dos padres. En este AG, se opta por la cruza en un punto. Sin embargo, deben formarse algunas consideraciones adicionales, principalmente por dos razones: primera, es indispensable asegurarse que los hijos resultantes son cadenas de adición válidas (o sea, las factibles) y en segundo lugar, los cromosomas son de longitud variable.

El operador de cruza adoptado tiene una característica muy peculiar, ya que la primera parte de la cadena del hijo uno toma los valores de la primera parte de la cadena del padre uno, pero la siguiente parte de la cadena del hijo uno se determina por medio de una regla, la cual trata de que la cadena de adición se completen con la configuración de la cadena del segundo padre, después del punto de cruza, como se muestra en la figura 6.

Por ejemplo, si tenemos un $e = 20$ un punto de cruza igual a 4 y los padres son los siguientes:

$Padre1 : 1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 20$

$Padre2 : 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 18 \rightarrow 20$

Entonces el hijo1 quedará de la siguiente manera:

$Hijo1 : 1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 12 \rightarrow 20$

Como podemos ver, los números $1 \rightarrow 2 \rightarrow 4$ son de la primera parte del Padre1 y los números $8 \rightarrow 12$ son de la configuración del Padre2, porque el Padre2 sumó $3 + 3$ para formar el siguiente número que es 6. Entonces, el hijo tiene que sumar $4 + 4$ para formar el siguiente número y para formar

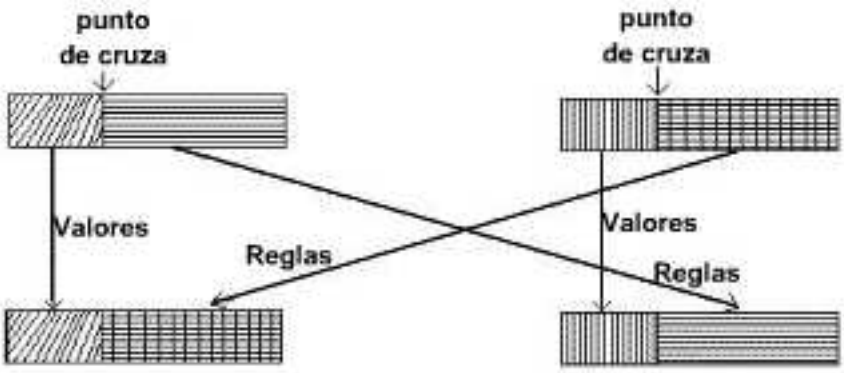

Figura 6: Representación del tipo de cruza utilizada.

el 9 el Padre2 utilizó el último y el penúltimo número. Ahora el hijo1 tiene que sumar el 8 y el 4 los cuales son el último y el penúltimo número, respectivamente lo que forma el número 12. Pero hay un pequeño problema con el siguiente número: el Padre2 sumo $9 + 9 = 18$, pero el Hijo1 no puede seguir con esa configuración porque si sumamos $12 + 12 = 24$ y ésta ya no sería una cadena de adición válida. Lo que se hace en este caso es completar la cadena de forma aleatoria. Para formar el hijo2 es similar.

El **Algoritmo 5.3** define el operador de cruza.

Algorithm 5.3: CruzaUtilizada(Pc)

for $i \leftarrow 1$ **to** $Tp/2$

$P1 = GeneraAleatorio(1, Tp)$

$P2 = GeneraAleatorio(1, Tp)$

comment: l es la longitud del cromosoma

$PuntoC = GeneraAleatorio(2, l-2)$

comment: Pasando la primera parte del padre al hijo 1

for $k \leftarrow 0$ **to** $PuntoC$

do $C1_k \leftarrow P1_k$

comment: Completando al hijo 1 con las reglas de la madre

for $k \leftarrow PuntoC + 1$ **to** $length$

comment: Busque valores a y b talles que:

$P2_k \leftarrow P2_a + P2_b$

comment: Entonces asigne:

$C1_k \leftarrow C1_a + C1_b$

comment: Pasando la primera parte de la madre al hijo 2

for $k \leftarrow 0$ **to** $PuntoC$

do $C2_k \leftarrow P2_k$

comment: Completando al hijo 2 con las reglas del padre

for $k \leftarrow PuntoC + 1$ **to** $length$

comment: Busque valores a y b talles que:

$P1_k \leftarrow P1_a + P1_b$

comment: Entonces asigne:

$C2_k \leftarrow C2_a + C2_b$

5.6. Tipo de Mutación

En este apartado se explica cómo se definió el operador de mutación para este algoritmo genético. Se hace notar que la definición de este operador nos permite introducir cambios aleatorios en el cromosoma, mientras preserve cadenas de adición validas.

El **algoritmo 5.4** ilustra el funcionamiento del operador de mutación.

Algorithm 5.4: MutacionUtilizada(Pm)

for $i = 1$ **to** Tp
- **comment:** l es la longitud del cromosoma
- $Punto = GeneraAleatorio(2, l-2)$
- **comment:** Empezar a mutar al individuo
- **for** $k = Punto + 1$ **to** l
  - **if** $Flip(Z)$
    - **then** $C_k \leftarrow 2C_{k-1}$
    - **else**
      - **if** $Flip(0{,}5)$
        - **then** $C_k \leftarrow C_{k-1} + C_{k-2}$
        - **else**
          - **comment:** m y n enteros, con $m, n \in [0, l-2]$
          - $C_k = C_m + C_n$

En este pseudocódigo hay una función llamada Flip(float i). Esta fun-

ción es de tipo booleana ya que decide si entra o no entra a la instrucción especificada. Esta función devuelve CIERTO con una probabilidad i, dada por el usuario.

En este operador de mutación hay tres tipos de cambios. El primer cambio consiste en duplicar el número más grande (el último número de la cadena ya calculado) de la cadena para hacer una aproximación mas rápida al objetivo. El segundo tipo consiste en sumar el último y el antepenúltimo número de la cadena de adición; este cambio también tiene una buena velocidad de aproximación. El último tipo de cambio consiste en encontrar dos números m, n de forma aleatoria; estos números tienen que estar forzosamente entre 0 y la posición del último número ya calculado de la cadena. Esto se debe de cumplir para que la cadena de adición siga siendo válida. Por ejemplo:

Objetivo = 30

Cad: $1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16$

En este caso, m y n tendría que estar en el intervalo [0,4] para poder calcular el siguiente número de la cadena de adición.

El operador de mutación debe ser aplicada a los hijos que resultaron de la cruza entre padres. Una vez aplicada la mutación, los individuos resultantes pasarán a formar la nueva población.

Los operadores de selección cruza y mutación vuelven a aplicar en cada generación (iteración) del AG, hasta cumplirse el criterio de paro (p.ej., un número máximo de generaciones).

Resumen

Es muy importante el tipo de representación para codificar las soluciones potenciales del problema de interés para un AG.

El tamaño de población que se vaya a asignar a un AG es importante porque la población le da la diversidad al AG. Entre más grande sea el tamaño de la población, mejores soluciones obtendrá el AG, pero hay que tener cuidado para que este valor no vaya a afectar la eficacia del AG, dado que mayores tamaños de población hacen más lenta la convergencia.

El tipo de representación, el tamaño de población, la función de aptitud, los operadores de selección, cruza y mutación son todos indispensables en un AG.

El porcentaje de cruza y de mutación son los que más afectan el desempeño de un AG. Al modificar estos valores, el AG suele variar sus soluciones, por lo que estos parámetros se suelen ajustar empíricamente, tras realizar numerosos experimentos.

En el capítulo siguiente se muestran los resultados que se obtuvieron por parte del AG presentado en este capítulo. Además se hace una comparación del AG con distintos métodos deterministas. También se calcula la cadena de adición mínima para algunos exponentes que dan pie a instancias muy difíciles de resolver con métodos deterministas.

Capítulo VI

Resultados

6. Resultados

En este capítulo se muestran los resultados que se obtuvieron con el algoritmo genético propuesto. Este algoritmo se implemento en C bajo el sistema operativo Linux.

6.1. Introducción

Para validar el desempeño del AG descrito en la sección anterior, se condujo una serie de experimentos, con el fin de comparar los resultados experimentales del AG en contra de los obtenidos usando varios métodos deterministas tradicionales.

La primera serie de experimentos consistió en encontrar las sumas de las cadenas de adición para todo e de exponentes en un intervalo dado. Luego, como una segunda prueba, se aplicó este algoritmo genético para una clase especial de exponentes cuyas cadenas óptimas de adición son particularmente duras de encontrar.

6.2. Comparando Resultados con Otros Algoritmos

Todos estos experimentos fueron realizados aplicando los parámetros siguiente del AG: El tamaño de población $N = 100$, Número de Generaciones = 300, Porcentaje de cruza $Pc = 0{,}6$, Porcentaje de mutación $Pm = 0{,}5$, Probabilidad $Z = 0{,}7$ (usado en el operador de mutación), Selección por Torneo. Todos los resultados estadísticos mostrados aquí se produjeron de 30 corridas independientes del algoritmo con semillas diferentes e indepen-

Valor óptimo=**10808**	
Estrategia	**Tamaño total**
Dyadic	10837
Total	10821
Fermat	10927
Dichotomic	11064
Factor	11088
Binary	11925
Quaternary	11479
Algoritmo Genético	Mejor: 10818 Promedio: 10824.07 Media: 10824 Peor: 10830 Desv. Estándar: 2.59

Tabla 1: Sumando longitudes de cadenas de adición para exponentes $e \in [1, 1000]$

dientes y aleatorias (adoptando una distribución uniforme).

Usando este AG descrito previamente, se calculan las sumas de las cadena de adición para todos los primeros 1000 exponentes, i.e. el $e \in [1, 1000]$. El valor acumulado obtenido fue comparado en contra de los valores de los siguientes métodos deterministas: El Dyadic, el de Total, el de Fermat, el de Dichotomic, el de Factor, el de Quaternary y Binary . Todos los resultados encontrados son mostrados en la tabla 1.

Puede verse que, en el mejor caso, el AG obtuvo un mejor resultado que los otros seis métodos. Es notable que ninguna de las estrategias deterministas pudieron encontrar el valor óptimo que se encontró realizando una búsqueda exhaustiva.

para todo $e \in [1, 512]$	para todo $e \in [1, 2000]$	para todo $e \in [1, 4096]$
Optimo: 4924	Optimo: 24063	Optimo: 54425
Binary: 5388	Binary: 26834	Binary: 61455
Quaternary: 5226	Quaternary: 25923	Quaternary: 58678
Algoritmo Genético	Algoritmo Genético	Algoritmo Genético
Mejor: 4925	Mejor: 24124	Mejor: 54648
Promedio: 4927.7	Promedio: 24135.17	Promedio: 54684.13
Media: 4927	Media: 24136	Media: 54685
Peor: 4952	Peor: 24144	Peor: 54709
Desv. Estándar: 4.74	Desv. Estándar: 5.65	Desv. Estándar: 13.55

Tabla 2: Sumando longitudes de cadenas de adición para 512, 2000 y 4096

Además, se calcularon las sumas de las cadenas de adición para todos los exponentes en los intervalos de $e \in [1, 512]$, $e \in [1, 2000]$ y $e \in [1, 4096]$. Los métodos usados fueron el AG, el método Binary y el método Quaternary. En todos los casos, para propósitos de comparación, se calcularon los valores óptimos correspondientes (obtenidos por numeración). Esos resultados son mostrados en la tabla 2.

Claramente, los resultados obtenidos por el AG funcionaron mejor que ambos, el método Binary y el método Quaternary, hasta en el peor caso. Podemos observar para los tres casos considerados (v.g. 512, 2000 y 4096), El AG obtuvo una buena aproximación del valor óptimo.

6.3. Resultados Optimos para Exponentes Especiales Duros de Optimizar

Sea $e = c(r)$ el exponente más pequeño que puede ser alcanzado usando una cadena de adición de r de largo. Las soluciones para esa clase de

exponentes se conocen hasta $r = 30$ y una recopilación de ellos puede ser encontrada en [13]. La dificultad computacional de encontrar la cadena de adición más pequeña para estos exponentes parece estar entre las más duras.

Se muestra las soluciones encontradas por el AG para la clase de exponentes indicados en la tabla 3 y la tabla 4. Es notable que en 24 de 30 exponentes, el acercamiento del AG pudiese encontrar la cadena más pequeña de adición. Sin embargo, para los 6 exponentes restantes (a saber, 357887, 1176431, 2211837,4169527, 7624319 y 14143037), esta el AG encontró la cadena adición una unidad arriba del óptimo.

exponente $e = c(r)$	Cadena de Adición	Longitud r
1	1	0
2	$1 \rightarrow 2$	1
3	$1 \rightarrow 2 \rightarrow 3$	2
5	$1 \rightarrow 2 \rightarrow 3 \rightarrow 5$	3
7	$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7$	4
11	$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 11$	5
19	$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 12 \rightarrow 19$	6
29	$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 11 \rightarrow 18 \rightarrow 29$	7
47	$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 10 \rightarrow 20 \rightarrow 40 \rightarrow 45 \rightarrow 47$	8
71	$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 12 \rightarrow 17 \rightarrow 34 \rightarrow 68 \rightarrow 71$	9
127	$1 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 48 \rightarrow 72 \rightarrow 120 \rightarrow 126 \rightarrow 127$	10
191	$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 10 \rightarrow 20 \rightarrow 21 \rightarrow 42 \rightarrow 63 \rightarrow 126 \rightarrow 189 \rightarrow 191$	11
379	$1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 10 \rightarrow 15 \rightarrow 25 \rightarrow 50 \rightarrow 75 \rightarrow 150 \rightarrow 300 \rightarrow 375 \rightarrow 379$	12
607	$1 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 48 \rightarrow 96 \rightarrow 192 \rightarrow 384 \rightarrow 576 \rightarrow 600 \rightarrow 606 \rightarrow 607$	13
1087	$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 18 \rightarrow 36 \rightarrow 74 \rightarrow 144 \rightarrow 216 \rightarrow 432 \rightarrow 864 \rightarrow 865 \rightarrow 1081 \rightarrow 1087$	14
1903	$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 10 \rightarrow 13 \rightarrow 26 \rightarrow 52 \rightarrow 104 \rightarrow 105 \rightarrow 210 \rightarrow 420 \rightarrow 840 \rightarrow 1680 \rightarrow 1890 \rightarrow 1903$	15
3583	$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 18 \rightarrow 36 \rightarrow 72 \rightarrow 108 \rightarrow 216 \rightarrow 432 \rightarrow 864 \rightarrow 1728 \rightarrow 3456 \rightarrow 3564 \rightarrow 3582 \rightarrow 3583$	16
6271	$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 48 \rightarrow 96 \rightarrow 192 \rightarrow 384 \rightarrow 768 \rightarrow 1536 \rightarrow 3072 \rightarrow 6144 \rightarrow 6240 \rightarrow 6264 \rightarrow 6270 \rightarrow 6271$	17
11231	$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 25 \rightarrow 50 \rightarrow 100 \rightarrow 200 \rightarrow 400 \rightarrow 800 \rightarrow 1600 \rightarrow 3200 \rightarrow 6400 \rightarrow 9600 \rightarrow 11200 \rightarrow 11225 \rightarrow 11231$	18
18287	$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 15 \rightarrow 30 \rightarrow 45 \rightarrow 47 \rightarrow 94 \rightarrow 188 \rightarrow 190 \rightarrow 380 \rightarrow 760 \rightarrow 1520 \rightarrow 3040 \rightarrow 6080 \rightarrow 12160 \rightarrow 18240 \rightarrow 18287$	19
34303	$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 14 \rightarrow 28 \rightarrow 56 \rightarrow 112 \rightarrow 224 \rightarrow 448 \rightarrow 504 \rightarrow 1008 \rightarrow 2016 \rightarrow 4032 \rightarrow 8064 \rightarrow 16128 \rightarrow 32256 \rightarrow 34272 \rightarrow 34300 \rightarrow 34303$	20
65131	$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 48 \rightarrow 72 \rightarrow 144 \rightarrow 288 \rightarrow 576 \rightarrow 1152 \rightarrow 2304 \rightarrow 4608 \rightarrow 4611 \rightarrow 9222 \rightarrow 18444 \rightarrow 27666 \rightarrow 55332 \rightarrow 55908 \rightarrow 65130 \rightarrow 65131$	21
110591	$\rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 10 \rightarrow 20 \rightarrow 40 \rightarrow 80 \rightarrow 160 \rightarrow 320 \rightarrow 640 \rightarrow 1280 \rightarrow 2560 \rightarrow 2570 \rightarrow 5140 \rightarrow 7710 \rightarrow 12850 \rightarrow 25700 \rightarrow 51400 \rightarrow 102800 \rightarrow 110510 \rightarrow 110590 \rightarrow 110591$	22
196591	$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 15 \rightarrow 30 \rightarrow 60 \rightarrow 120 \rightarrow 240 \rightarrow 480 \rightarrow 720 \rightarrow 1440 \rightarrow 2880 \rightarrow 5760 \rightarrow 11520 \rightarrow 23040 \rightarrow 46080 \rightarrow 92160 \rightarrow 184320 \rightarrow 19584 \rightarrow 196560 \rightarrow 196590 \rightarrow 196591$	23

Tabla 3: Las cadenas de adición más pequeñas para una clase especial de exponentes

exponente $e = c(r)$	Cadena de Adición	Longitud r
357887	$1 \to 2 \to 3 \to 4 \to 8 \to 16 \to 32 \to 64 \to 128 \to 256 \to 257 \to 514 \to 771 \to 11542$ $\to 3084 \to 6168 \to 12336 \to 24672 \to 49344 \to 49347 \to 98691 \to 148038 \to 296076$ $\to 345423 \to 357759 \to 357887$	24
685951	$1 \to 2 \to 4 \to 6 \to 7 \to 14 \to 21 \to 42 \to 84 \to 168 \to 336 \to 504 \to 840 \to 1680$ $\to 3360 \to 6720 \to 13440 \to 26880 \to 53760 \to 57120 \to 114240 \to 228480$ $\to 342720 \to 685440 \to 685944 \to 685951$	25
1176431	$1 \to 2 \to 4 \to 5 \to 10 \to 15 \to 19 \to 38 \to 76 \to 152 \to 304 \to 608 \to 612$ $\to 1224 \to 2448 \to 4896 \to 9792 \to 19584 \to 29376 \to 58752 \to 117504 \to 235008$ $\to 352512 \to 587520 \to 1175040 \to 1176264 \to 1176416 \to 1176431$	27
2211837	$1 \to 2 \to 3 \to 6 \to 9 \to 15 \to 30 \to 60 \to 120 \to 126 \to 252 \to 504 \to 1008$ $\to 2016 \to 4032 \to 8062 \to 16128 \to 16143 \to 32286 \to 64572 \to 129144 \to 258288$ $\to 516576 \to 1033152 \to 2066304 \to 2195448 \to 2211591 \to 2211717 \to 2211837$	28
4169527	$1 \to 2 \to 3 \to 6 \to 12 \to 24 \to 48 \to 96 \to 192 \to 384 \to 768 \to 1536 \to 2304$ $\to 4608 \to 9216 \to 18432 \to 36864 \to 73728 \to 147456 \to 294912 \to 589824 \to 589825$ $\to 1179650 \to 1769475 \to 3538950 \to 4128775 \to 4165639$ $\to 414167943 \to 4169479 \to 4169527$	29
7624319	$1 \to 2 \to 3 \to 6 \to 12 \to 18 \to 36 \to 72 \to 144 \to 288 \to 576 \to 1152 \to 1224$ $\to 2448 \to 4896 \to 9792 \to 19584 \to 39168 \to 78336 \to 156672 \to 313344$ $\to 626688 \to 1253376 \to 1254600 \to 1274184 \to 1274185 \to 2548370$ $\to 5096740 \to 6370925 \to 7624301 \to 7624319$	30
14143037	$1 \to 2 \to 3 \to 6 \to 12 \to 18 \to 30 \to 60 \to 120 \to 240 \to 480 \to 960 \to 961$ $\to 1922 \to 3844 \to 7688 \to 11532 \to 23064 \to 46128 \to 92256 \to 184512 \to 369024$ $\to 461280 \to 830304 \to 1660608 \to 3321216 \to 6642432$ $\to 13284864 \to 14115168 \to 14138232 \to 14142076 \to 14143037$	31

Tabla 4: Las cadenas de adición más pequeñas para una clase especial de exponentes (Complemento)

Resumen

En este capítulo se reflejaron los resultados que se obtuvieron con las corridas del programa, que implementa el AG.

Se hace una validación del acercamiento del AG y además se conduce una serie de experimentos, con el fin de comparar los resultados experimentales del AG contra los obtenidos usando varios métodos deterministas tradicionales.

Se aplica este AG para una clase especial de exponentes cuyas cadenas óptimas de adición son particularmente duras de encontrar.

Capítulo VII

Conclusiones y Trabajo Futuro

7. Conclusiones y Trabajo Futuro

En este escrito se describe cómo un algoritmo genético puede ser aplicado para el problema de calcular la cadena de adición óptima más pequeña de exponentes.

El AG presentado en este trabajo fue capaz de encontrar casi todas las cadenas de adición óptima para cualquier e fijo dado del exponente con $e < 4096$.

Teniendo en cuenta el valor óptimo (que fue encontrado por enumeración) el error porcentual de nuestra estrategia de AG estaba dentro de 0.4 % del óptimo para todos los casos considerados. En otras palabras, para cualquier e fijo dado del exponente e, con $e < 4096$, nuestra estrategia pudo encontrar la cadena de adición más pequeña en un 99.6 % de los casos.

En un segundo experimento que pretendía evaluar la potencia del AG como un motor de búsqueda, lo probamos para generar las cadenas de adición más pequeñas de una clase de exponentes particularmente duros para optimizar, cuya longitud óptima es conocida para los primeros 30 miembros de la familia. En muchos de estos casos considerados, el AG pudo encontrar los valores óptimos.

Algunas de las extensiones o trabajos futuros que podrían realizarse sobre el AG propuesto son las siguientes:

- El algoritmo descrito hace una selección mediante torneo, este operador de selección suele elevar la presición de selección, lo cual puede producir convergencia prematura en algunos casos. Sería interesante utilizar otras técnicas de selección tales como las proporcionales (p.ej. la ruleta).

- También se utiliza el operador de cruza en un punto, debido a que la representación de los cromosomas es entera y entre más puntos tenga la cruza, más difícil es conseguir cromosomas válidos. Sin embargo, es deseable disceñar otros operadores de cruza que emulen el efecto de la cruza uniforme, sin generar demasiadas soluciones inválidas.

- La representación adoptada por el AG es de tipo entera, pero sería interesante probar con una representación binaria.

Referencias

[1] John Jenkins, editor: *Genetics*, Houghton Mifflin Company, Boston, Massachusetts, 1984

[2] Jean Baptiste Lamarck, editor. *Zoological Philosophy:* An Exposition with Regard to the Natural History of Animals. Chicago University Press, Chicago, 1984 (Publicado originalmente en 1809 en francés, como Philosophie Zoologique).

[3] August Weismann, editor. *The Germ Plasm:* A Theory of Heredity. Scott, London, UK, 1893.

[4] Charles Darwin. *The Origin of Species by Means of Natural Selection or the Preservation of Favored Races in the Struggle for Life.* The Book League of America, 1929. Originally published in 1859.

[5] Hans-Paul Schwefel, editor. *Evolution and Optimization Seeking.* John Wiley & Sons, New York, 1995.

[6] Lawrence J. Fogel *Intelligence Through Simulated Evolution. Forty years of Evolutionary Programming.* John Wiley & Sons, New York, 1999.

[7] http://geneura.ugr.es/∼jmerelo/ie/intro.htm (11/10/2005)

[8] http://www.orcero.org/irbis (15/10/2005)

[9] http://www.cs.cinvestav.mx/∼ EVOCINV/tutorials/computacionevolutiva.htm (20/10/2005)

[10] http://www.pcai.com/web/ai_info/genetic_algorithms.html &prev=/search%3Fq%3DGenetics%2Balgorithms%26hl%3Des%26lr%3D (11/10/2005)

[11] F. Bergeron, J. Berstel, and S. *Brlek. Efficient computation of addition chains.* Journal de thorie des nombres de Bordeaux, 6:21-38, 1994.

[12] Donald Ervin Knuth. *The Art of Computer Programming 3rd. ed.* Addison-Wesley,

[13] J. Bos and M. Coster. *Addition chain heuristics.* In G. Brassard, (editor) Advances in Cryptology —CRYPTO 89 *Lecture Notes in Computer Science*, 435:400-407, 1989. D.

[14] Bleinchenbacher and A. Flammenkamp. *An Efficient Algorithm for Computing Shortest Addition Chains*, 1997.

[15] D. M. Gordon. *A survey of fast exponentiation methods.* Journal of Algorithms, 27(1):129-146, April 1998.

[16] A. J. Menezes, Paul C. van Oorschot, and Scott A.Vanstone. *Handbook of Applied Cryptography. CRC Press*, Boca Raton, Florida, 1996.

[17] E. Baldwin. *Genética Elemental.* Limusa, 1983. Algorithms, 27(1):129-146, April 1998.

[18] Jean Baptiste Lamarck, editor. *Zoological Philosophy: An Exposition with Regard to the Natural History of Animals.* Chicago University Press, Chica- go, 1984 (Publicado originalmente en 1809 en francés, como Philosophie Zooologique).

[19] A. Wetzel. *Evaluation of the effectiveness of genetic algorithms in combinational optimization*. University of Pittsburgh, Pittsburgh, 1983.

[20] A. Brindle. *Genetic Algorithms for Function Optimization*. Tesis doctoral , Department of Computer Science, University of Alberta, Edmonton, Alberta, 1981.

[21] David B. Fogel. **Evolutionary Computation**. Toward a New Philosophy of Machine Intelligence. The Institute of Electrical and Electronic Engineers, New York, 1995.

Apéndice

A. Terminología utilizada en computación evolutiva

- **Aptitud**: Valor que se asigna a cada individuo y que indica qué tan bueno es éste con respecto a las demás soluciones de un problema.
- **Cromosoma**: Estructura de datos que contiene una cadena de parámetros de diseño o genes. Por ejemplo, puede ser una cadena de bits.
- **Cruza**: Es un operador que forma un nuevo cromosoma combinando partes de cada uno de los cromosomas padres.
- **Elitismo**: Mecanismo utilizado en algunos algoritmos evolutivos para asegurar que los cromosomas de los miembros más aptos de una población pasen a la siguiente generación sin ser alterados.
- **Gene**: Es una subsección de un cromosoma. Usualmente codifica el valor de un solo parámetro (variable de decisión).
- **Generación**: Es una iteración que involucra la creación de una nueva población por medio de los operadores básicos de reproducción.
- **Individuo**: Un solo miembro de la población de soluciones potenciales a un problema. Cada individuo contiene un cromosoma que representa una solución posible al problema.
- **Mutación**: Operador que forma un nuevo cromosoma a través de alteraciones a los genes del cromosoma.

- **Genotipo**: Se denomina genotipo a la codificación (por ejemplo binaria) de los parámetros que representan una solución del problema a resolverse.

- **Fenotipo**: Se denomina fenotipo a la decodificación del cromosoma. Es decir, a los valores obtenidos al pasar de la representación (binaria) a la usada por la función objetivo.

B. Código para Generar la Población Inicial

El siguiente código genera la población inicial.

Algoritmo 1.

```
void genpoblacionIni()
{
   int inicio,fin;
   int i,j;
   clrscr();
   randomize();
/**Se rellena la matriz de 1 y de 2 la primera y segunda columna
**/
   for( i=0;i<2;i++)
   {
     for( j=0;j<Tpob;j++)
     {
       poblacion[j][i]=i+1;
     }
   }
//*********************************************************************
   for(j=0;j<Tpob;j++)
   {
     for(i=2;i<150;i++)
     {
       poblacion[j][i]=NumCorrectocopia(j,i);
```

```
        if( poblacion[j][i]==ob || i==149)
        {
           aptitud[j]=i+1;
           break;
        }
      }
   }
}
```

Fin de algoritmo 1

C. Código que Implementa la Selección de Individuos

Algoritmo 2.

```
void selección_Torneo()
{
   int i,k,j,Comp1,Comp2,mejor=0;
   //elitismo
   clrscr();
   for(i=1;i<T_pob;i++)
   {
     if(aptitud[mejor]>aptitud[i])
     {
       mejor=i;
     }
   }
   for(k=0;k<aptitud[mejor];k++)
   {
     Padres[0][k]=poblacion[mejor][k];
   }
   aptitud_Pa[0]=aptitud[mejor];
//********
   randomize();
   for(i=1;i<T_pob;i++)
```

```
{
  Comp1=(random(T_pob));
  Comp2=(random(T_pob));
  if(aptitud[Comp1]<aptitud[Comp2])
  {
    for(j=0;j<aptitud[Comp1];j++)
    {
      Padres[i][j]=poblacion[Comp1][j];
    }
    aptitud_Pa[i]=aptitud[Comp1];
  }
  else if(aptitud[Comp1]>aptitud[Comp2])
  {
    for(j=0;j<aptitud[Comp2];j++)
    {
      Padres[i][j]=poblacion[Comp2][j];
    }
    aptitud_Pa[i]=aptitud[Comp2];
  }
  else if(aptitud[Comp1]==aptitud[Comp2])
  {
    for(j=0;j<aptitud[Comp2];j++)
    {
      Padres[i][j]=poblacion[Comp2][j];
    }
```

```
            aptitud_Pa[i]=aptitud[Comp2];
        }
    }
}
```

Fin de Algoritmo 2.

D. Código que Realiza el Operador de Cruza

El código que se muestra a continuación implementa el operador de cruza.

Algoritmo 3:

```
void cruza(float por_cr)
{
   int punto,P1,P2,i,j,r,k,Correc,ban;
   float num;
   randomize();
// ELITISMO*****************************
   for(r=0;r<aptitud[0];r++)
   {
          Padres[0][r]=poblacion[0][r];
   }
   aptitud_Pa[0]=aptitud[0];
   for(r=0;r<aptitud[0];r++)
   {
          Padres[1][r]=poblacion[0][r];
   }
   aptitud_Pa[1]=aptitud[0];
//FIN DE ELITISMO**********************
   for(i=2;i<T_pob;i+=2)
   {
     P1=random(T_pob);
```

```
P2=random(T_pob);
num=(float(random(101))/100.0);
if(num <por_cr && (aptitud[P1]>2 && aptitud[P2]>2))
{
  if(aptitud[P1]<aptitud[P2])
  {
    punto= random(aptitud[P1]-2)+1;
  }
  else if(aptitud[P1]>aptitud[P2])
  {
    punto=random(aptitud[P2]-2)+1;
  }
  else
  {
    punto=random(aptitud[P2]-2)+1;
  }
  //copia la primera parte del hijo 1 y 2
  tem=punto;
  for(j=0;j<punto;j++)
  {
    Padres[i][j]=poblacion[P1][j];
    Padres[i+1][j]=poblacion[P2][j];
  }
  for(k=punto;k<150;k++)
  {
```

```
pad=P2;
if(k<aptitud[P2] && Padres[i][k-1]<ob)
{
   Correc=Padres[i][BuscaNumCruza(k,P2)]+Padres[i][k-1];
   if(Correc<=ob)
   {
      Padres[i][k]=Correc;
   }
   else
   {
      Padres[i][k]=NumCorrecto_mut_cru(i,k);
   }
   if(Padres[i][k]==ob || k==149)
   {
      aptitud_Pa[i]=k+1;
      break;
   }
}
else
{
   Padres[i][k]=NumCorrecto_mut_cru(i,k);
   if(Padres[i][k]==ob || k==149)
                  aptitud_Pa[i]=k+1;
      break;
   }
```

```
    }
  }
  for(j=punto;j<150;j++)
  {
    pad=P1;
    if(j<aptitud[P1] && Padres[i+1][j-1]<ob)
    {
      Correc=Padres[i+1][BuscaNumCruza(j,P1)]+Padres[i+1][j-1];
      if(Correc<=ob)
      {
        Padres[i+1][j]=Correc;
      }
      else
      {
        Padres[i+1][j]=NumCorrecto_mut_cru(i+1,j);
      }
      Padres[i+1][BuscaNumCruza(j,P1)]+Padres[i+1][j-1];
      if(Padres[i+1][j]==ob || j==149)
      {
        aptitud_Pa[i+1]=j+1;
        break;
      }
    }
    else
    {
```

```
            Padres[i+1][j]=NumCorrecto_mut_cru(i+1,j);
            if(Padres[i+1][j]==ob || j==149)
            {
              aptitud_Pa[i+1]=j+1;
              break;
            }
          }
        }
        else
        {
          for(j=0;j<aptitud[P1];j++)
          {
            Padres[i][j]=poblacion[P1][j];
          }
          aptitud_Pa[i]=aptitud[P1];
          for(j=0;j<aptitud[P2];j++)
          {
            Padres[i+1][j]=poblacion[P2][j];
          }
          aptitud_Pa[i+1]= aptitud[P2];
        }
    }
}
```

Fin de Algoritmo 3

E. Código que Implementa el Operador de Mutación

Algoritmo 4.

```
void mutacion(float por_mut)
   {
   float num,porc;
   int punto,i,j,r;
   randomize();
// ELITISMO*****************************
   for(j=0;j<aptitud[0];j++)
   {
      Padres[0][j]=poblacion[0][j];
   }
   aptitud_Pa[0]=aptitud[0];
//FIN DE ELITISMO**********************
   for(i=1;i<T_pob;i++)
   {
      num=(float(random(101))/100.0);
      if(num <= por_mut && aptitud[i]>2)
      {
         punto=random(aptitud[i]-2)+1;
         for(r=0;r<punto;r++)
         {
            Padres[i][r]=poblacion[i][r];
```

```
        }
        for(j=punto;j<150;j++)
        {
          Padres[i][j]= NumCorrect_mut_cru(i,j);
          if(Padres[i][j]==ob || j==149)
          {
            aptitud_Pa[i]=j+1;
            break;
          }
        }
      }
      else
      {
        for(j=0;j<aptitud[i];j++)
        {
          Padres[i][j]=poblacion[i][j];
        }
        aptitud_Pa[i]=aptitud[i];
      }
    }
    textbackground(BLUE);
}
```

Fin de Algoritmo 4

Índice de Algoritmos