Algoritmos Genéticos

Ángel Kuri Morales
Centro de Investigación en Computación,
Instituto Politécnico Nacional.

José Galaviz Casas
Facultad de Ciencias,
Universidad Nacional Autónoma de México.

PREFACIO

Los algoritmos genéticos en particular y la computación evolutiva en general, se han constituido en uno de los focos de mayor atracción para investigadores de diversas ramas de la computación y la ingeniería, para constatarlo basta ver el gran número de artículos que, sobre el tema y sus aplicaciones, han aparecido en los últimos años. Los algoritmos genéticos son, sin duda, un importante punto de partida para lo que será el desarrollo de la computación en el siglo XXI.

Los algoritmos genéticos son métodos heurísticos de búsqueda inspirados en lo que sabemos acerca del proceso de la evolución natural. Si la naturaleza ha sido capaz de generar organismos óptimos para desempeñarse en medios ambientes sumamente complejos, por qué no copiar sus métodos para resolver nuestros propios problemas y tratar de encontrarles soluciones que, de alguna manera, sean óptimas. La gran popularidad que han alcanzado los algoritmos genéticos, así como otras técnicas heurísticas, se debe, en buena parte, a que hacen posible abordar problemas en los que es muy difícil aplicar procedimientos matemáticos tradicionales. La gran plasticidad inherente a estos métodos, hace posible aplicarlos a la solución de muy diversos problemas sin hacer modificaciones sustanciales al procedimiento general. Es decir, no suponen nada o casi nada acerca del problema a resolver, a diferencia de los métodos tradicionales en los que se exige que el modelo matemático del problema consista de una función claramente definida y con ciertas propiedades.

Este libro ofrece al lector desde una introducción detallada a los algoritmos genéticos, hasta la presentación de algunos temas de investigación de frontera en el tema. El contenido ha sido planeado para cursos de nivel licenciatura en carreras técnicas o científicas relacionadas con la computación. Para ello, el lector deberá tener conocimientos elementales de probabilidad, estadística, cálculo y estructuras de datos, similares a los que proveen los cursos de los primeros semestres del tipo de carreras mencionadas. No se requiere conocimiento previo acerca del tema del libro, sin embargo, para el estudiante con conocimientos de algoritmos genéticos, así como para el profesional interesado en el área, el libro resultará útil como material de referencia o introducción a algunos temas de investigación.

El primer capítulo introduce al lector al tema de los algoritmos genéticos, y establece aquellos elementos de la genética y la evolución natural que pretenden simular. Muestra un tipo particular de algoritmo genético conocido como *simple* e ilustra su funcionamiento a través de un ejemplo de optimización. Presenta luego un catálogo (el cual más que exhaustivo pretende ser representativo) de algunos otros métodos, tanto tradicionales como heurísticos, para resolver problemas de optimización. En el segundo capítulo se presentan de manera sumamente clara los fundamentos matemáticos de los algoritmos genéticos y se utilizan luego para investigar a fondo cómo es que éstos funcionan y cómo es que en algunos casos no lo hacen. El tercer capítulo utiliza lo visto en los dos capítulos

previos para plantear diferentes alternativas al modelo del algoritmo genético simple y propone nuevos problemas; además, prepara al estudiante para los dos capítulos siguientes. En el capítulo cuatro, tomando como base el algoritmo genético simple, lo analizado en el capítulo dos, y el llamado *algoritmo genético idealizado*, se construye un nuevo tipo de algoritmo genético, que hemos denominado *ecléctico*, que incorpora características que no poseía el simple y que mejora notablemente el desempeño, además de hacerlo mucho más flexible y robusto que su predecesor. Finalmente, en el capítulo cinco se presentan los algoritmos genéticos como herramientas para lograr aprendizaje de máquina en cuatro diferentes casos: coevolución, autómatas genéticos, sistemas clasificadores y ajuste paramétrico de orden libre para aproximaciones polinomiales multivariadas de datos experimentales.

El libro ha sido planeado para servir como texto en cursos avanzados sobre el tema a nivel licenciatura y como tal contiene ejercicios destinados a reforzar y ampliar los conocimientos adquiridos en el texto. Al final de cada capítulo se encuentra una serie de ejercicios teóricos cuyos objetivos fundamentales son: lograr una cabal comprensión de los conceptos contenidos en el capítulo, propiciar la reflexión individual o en grupo acerca de temas colaterales a los del texto y estimular la imaginación del ejecutante. Tras los ejercicios teóricos sigue una serie de ejercicios de programación que enfrentarán al estudiante con los problemas típicos de la implantación de algoritmos genéticos y lo capacitarán para elaborar sus propias innovaciones y explorar aquellas alternativas que le sean dictadas por su imaginación.

Los ejercicios del libro se han dividido en dos grupos: los que están y los que no. Se recomienda que el estudiante haga los del primer grupo. El instructor que utilice este libro podrá percatarse de que, en general, los estudiantes pasan por diversas etapas al enfrentarse a los ejercicios: curiosidad (¿cómo se les ocurrió esto a los autores?); negación (¡no puede ser! ¿de veras hay que hacerlo?); resignación (bueno ... trataré), y al final, en una espiral dialéctica, regresan a la actitud de curiosidad en un plano superior (¡vaya! a fin de cuentas no estuvo tan difícil, pero ¿qué pasaría si ...?). Luego los estudiantes comienzan a hacer los ejercicios del segundo grupo.

El lector puede enviar sugerencias, preguntas o contribuir a la fe de erratas del libro, escribiendo a una de las direcciones electrónicas siguientes (su participación será bienvenida):

jgc@fciencias.unam.mx akuri@cic.ipn.mx

Agradecemos a nuestras respectivas instituciones: el Centro de Investigación en Computación del Instituto Politécnico Nacional, el Instituto de Matemáticas y la Facultad de Ciencias de la Universidad Nacional Autónoma de México, por las facilidades que nos proporcionaron para la realización de esta obra. Agradecemos también a la Dirección de Publicaciones y Materiales Educativos del IPN y a Carlos Vizcaíno Sahagún por su invaluable apoyo en la edición. Damos las

gracias a nuestros estudiantes, que sirvieron de conejillos de indias y víctimas de nuestros experimentos en el uso del texto, por sus críticas, sugerencias y abnegación. Hay que agradecer también a los concienzudos revisores del texto, sus críticas a las versiones preliminares nos fueron de gran valor: no nos es posible agradecerles lo suficiente. Finalmente agradecemos también a nuestras familias, sin cuya condescendiente actitud hacia nuestra neurosis no hubiera sido posible la realización de esta obra.

Ángel Kuri Morales

José Galaviz Casas

I. INTRODUCCIÓN

I.1 LA NATURALEZA COMO OPTIMIZADORA

"AUNQUE el ingenio humano puede lograr infinidad de inventos, nunca ideará ninguno mejor, más sencillo y directo que los que hace la naturaleza, ya que en sus inventos no falta nada y nada es superfluo", escribió Leonardo Da Vinci (*Cuaderno de Notas, la vida y estructura de las cosas, el embrión*). Un siglo después, Johannes Goldschmidth, mejor conocido por su nombre latino *Fabricius ab Aquapendente* (quien se disputara con Galileo el descubrimiento de los defectos solares), afirmaría, anticipándose a Darwin: "La naturaleza perpetúa aquello que resulta mejor." (*De Motu Locali Animalium*). Todos los seres vivos que habitamos en este planeta somos, de alguna manera, obras casi perfectas. Cuando nos percatamos de la vasta complejidad que somos, de cómo nuestro cuerpo es una enredada madeja de relaciones delicadamente establecidas entre los distintos órganos, no podemos más que maravillarnos y otro tanto ocurre cuando percibimos al resto de los seres vivos de la misma manera. Recientemente, un grupo de ingenieros que diseñaba un submarino se percató de que el pingüino posee una forma tal, que al fluir el agua a su alrededor, prácticamente no genera turbulencias en las que se pierda la energía invertida por el ave al nadar y al mismo tiempo, posee un volumen cercano al máximo, lo que permite al pingüino acumular gran cantidad de grasa que lo aísla del frío exterior y constituye su reserva de energía. Se podrían citar miles de ejemplos más: "[...] cuanto más informados e iluminados estemos acerca de las obras de Dios, más inclinados estaremos a encontrarlas excelentes y totalmente conformes a cuanto se hubiera podido desear", diría Leibniz (*Discours de Metaphysique*).

La naturaleza (o Dios, como prefiere Leibniz) genera seres óptimos, seres perfectamente adaptados a su entorno, constituido por una infinidad de circunstancias: temperatura, presión atmosférica, precipitación, velocidad del viento, altitud, nivel de insolación, depredadores, alimentos, etc. En función de su entorno el pingüino es un animal perfecto para vivir en las soledades antárticas; en el Amazonas perecería. Su adaptación es perfecta porque, a lo largo de miles de generaciones se ha perfeccionado a pequeños saltos infinitesimales. Lo que vemos hoy en día es el resultado acumulado de miles de experimentos exitosos que han ido refinando paulatinamente alguna creación primigenia. Los experimentos fallidos, probablemente algunos órdenes de magnitud más numerosos que los exitosos, no los vemos. Los individuos resultantes perecieron compitiendo al lado de otros más aptos para sobrevivir. "La selección natural obra solamente mediante la conservación y acumulación de pequeñas modificaciones heredadas,

provechosas todas al ser conservado.", escribió Darwin y dio nombre a este proceso (*El origen de las especies, selección natural*).

Estas "modificaciones heredadas", señaladas por Darwin como las generadoras de organismos mejores, son llamadas mutaciones hoy en día y constituyen el motor de la evolución. Un organismo mutante ha sufrido una modificación que lo hace diferente al resto de sus congéneres. Esta modificación puede ser un inconveniente para él (la falta de algún miembro útil de su cuerpo, por ejemplo), pero puede ocurrir también que le confiera alguna cualidad que le permita sobrevivir más fácilmente que al resto de los individuos de su especie. Este organismo tendrá mayor probabilidad de reproducirse y heredará a sus descendientes la característica que le dio ventaja. Con el tiempo, gracias a la competencia, los organismos que en un principio eran raros se volverán comunes a costa de la desaparición del "modelo anterior". Se habrá dado entonces un paso en el proceso evolutivo.

Esta cualidad del proceso natural de la evolución (generar organismos óptimos sobre los que influyen infinidad de variables), llamó la atención de algunos científicos en las décadas de los cincuenta y sesenta. Un alemán, de apellido Rechenberg, introdujo en 1965 lo que denominó *evolutionsstrategie*, o estrategias evolutivas, como un método para optimizar funciones de varias variables que definían dispositivos tales como perfiles de alas de avión. En 1966 los investigadores Fogel, Owens y Walsh se dieron a la tarea de dejar evolucionar máquinas de estados finitos sometiendo sus diagramas de transición a cambios aleatorios (mutaciones), creando con ello lo que se conoce como *programación evolutiva*. También en los sesenta, John Holland, junto con algunos de sus colegas y alumnos de la Universidad de Michigan, desarrolló lo que se conoce como *algoritmos genéticos* (AGs). Sin embargo, el objetivo de Holland no era inventar un método de optimización basado en los mecanismos de la evolución natural, sino elaborar un estudio formal acerca del fenómeno de adaptación en sistemas naturales y artificiales, es decir, definir un modelo para la adaptación. El algoritmo genético definido por Holland en su trabajo [7] es un intento de abstraer las características esenciales del proceso evolutivo tal como se observa en la naturaleza, con vistas a importarlo a sistemas de cómputo.

En la actualidad, los AGs son preferentemente utilizados como métodos de búsqueda de soluciones óptimas que simulan la evolución natural y han sido usados con éxito en la solución de problemas de optimización combinatoria, optimización de funciones reales y como mecanismos de aprendizaje de máquina (*machine learning)*. Esto último les ha ganado un lugar en el campo de la inteligencia artificial.

I.2 UN POCO DE BIOLOGÍA

En esta sección se presentará superficialmente lo que se sabe acerca de los mecanismos de la herencia y la terminología utilizada en esta área. El objetivo es evidenciar la analogía entre los algoritmos genéticos y aquello que pretenden simular. El lector interesado en profundizar puede consultar [13].

Cada individuo de cada una de las especies que habitan en nuestro planeta poseen ciertas características que lo identifican. Si hablamos de seres humanos, cada uno posee cierta estatura, cierto color de ojos y de cabello y cierto tipo sanguíneo, entre otras muchas. Estas características "externas", aunque algunas de ellas no se puedan ver, como el tipo sanguíneo, constituyen lo que se denomina el *fenotipo* de un individuo. Cada una de estas características es igual a la correspondiente de alguno de los antecesores del individuo, es decir, nos son dadas por herencia, o por lo menos nos es dada cierta predisposición a ella (como la diabetes, por ejemplo). El fenotipo es resultado de la interacción del medio ambiente en que se desarrolla un individuo y la herencia que éste recibe de sus ancestros. La herencia impone ciertos límites o predisposiciones que, al sumarse con el medio, generan el fenotipo. A veces el medio no importa, por ejemplo, no puede intervenir en nuestro color de ojos, pero en otras influye de manera determinante. Si se posee cierta predisposición a padecer enfermedades cardiovasculares pero se tiene una excelente condición aeróbica desde pequeño, posiblemente éstas nunca se padezcan. El fenotipo de cada individuo está determinado por las proteínas que produce, y esto a su vez está definido en la información genética de cada una de sus células.

La información acerca de cuáles proteínas se producirán está contenida en los cromosomas del individuo. En cada célula somática (aquellas que constituyen el organismo) existen dos juegos de cromosomas que definen las mismas características; un juego es aportación del padre del individuo y el otro lo es de la madre. Un ser humano posee 23 pares de cromosomas.

En términos sencillos, un cromosoma es una larga molécula de ADN (Ácido Desoxirribonucleico), formada por cuatro distintos compuestos más simples llamados bases o nucleótidos: adenina (A), guanina (G), citosina (C) y timina (T). Cada subcadena de tres nucleótidos codifica un aminoácido diferente que al unirse con los generados por otros tercetos, formará una proteína. A las subcadenas de tres nucleótidos se les llama *codones* y al conjunto de nucleótidos que codifican una proteína completa se les llama *genes*. El valor que posee un gene determinado se denomina *alelo.*

Las células que poseen dos juegos de cromosomas se denominan *diploides*, ya que en éstas los cromosomas se encuentran en pares; también los genes deben estar en pares homólogos. Si ambos tienen el mismo alelo se dice que son *homocigos*, si no, son *heterocigos*, y en este último caso sólo uno de los alelos se manifestará en el fenotipo (éste se denomina *dominante*, su homólogo que no se manifiesta, se llama *recesivo*). El conjunto de todos los cromosomas, es decir,

toda la información genética de un individuo se llama *genoma* y el conjunto de genes contenidos en el genoma *genotipo*. Es éste el que determina, en buena medida, el fenotipo del individuo.

Hay unas células especiales llamadas *gametos*, que intervienen en la reproducción (los espermatozoides y los óvulos humanos pertenecen a esta categoría). Los gametos no se reproducen por *mitosis* como el resto de las células, el proceso de división se llama en este caso *meiosis*. En la mitosis las células producidas son diploides, mientras que en la meiosis el resultado, los gametos, son *haploides*, sólo tienen un juego de cromosomas.

Partiendo de una sola célula diploide el proceso meiótico es como sigue (figura F.I.2.A):

1. Se duplica el número de cromosomas en la célula, esto es, se hace una copia de cada cromosoma. Al final quedan dos juegos correspondientes al padre y dos a la madre.
2. Se cruzan un juego de cromosomas del padre con uno de la madre, formándose dos juegos de cromosomas híbridos. El resultado es un juego de cromosomas puros del padre, un juego puro de la madre y dos juegos de cromosomas híbridos.
3. Se divide la célula dos veces y al final del proceso quedan cuatro células haploides: una con cromosomas puros del padre, una con cromosomas puros de la madre y dos con cromosomas híbridos.

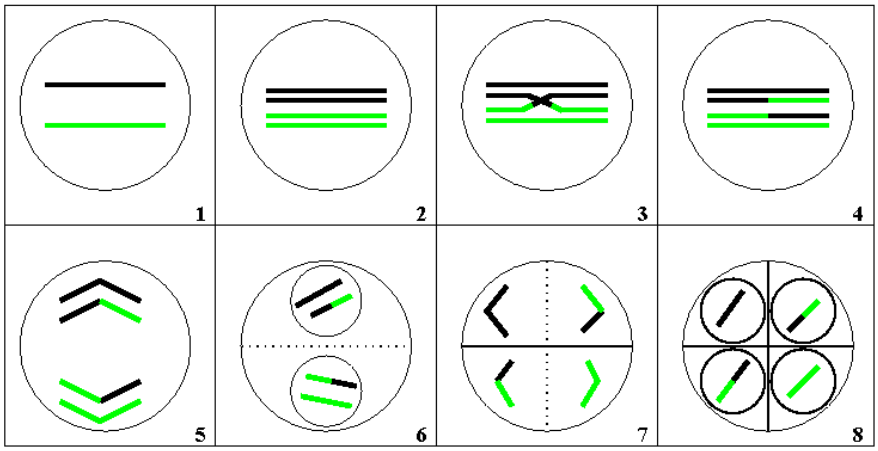


FIGURA F.I.2.A: *Reproducción celular por meiosis: En 1 aparece la célula diploide original, en 2 los cromosomas se replican, en 3 se forman quiasmas, en 4 ya hay cuatro juegos de cromosomas (del padre, de la madre y dos híbridos), en 5 se separan en pares, en 6 se forma una primera membrana celular, en 7 se separan nuevamente los cromosomas, finalmente en 8 se forma una nueva membrana que separa los nuevos núcleos (ahora cada uno tiene sólo un juego de cromosomas), cada nueva célula es haploide.*

En el paso dos del proceso de meiosis se mezclan las características del padre y la madre. Para el cruzamiento de dos cromosomas se forman entre ellos puntos de ruptura y unión de las cadenas de ADN. Estos puntos, llamados *quiasmas*, cortan el cromosoma en segmentos llamados *cromátidas* y unen cromátidas complementarias de dos distintos cromosomas. Al final cada cromosoma que participó en la cruza queda constituido por segmentos que ya poseía y por otros que eran de su análogo (figura F.I.2.B).

En el paso uno del proceso se deben replicar los cromosomas existentes. Hay una enzima encargada de copiarlos, ésta es la *ADN-polimerasa*. La molécula de ADN tiene forma de una doble hélice, como una escalera de caracol. La enzima abre por en medio los "escalones" de esta hélice y ensambla en cada mitad los nucleótidos que debe ensamblar (figura F.I.2.C). Ocasionalmente esta enzima comete un error, que puede ser causado por radiaciones energéticas externas o sustancias extrañas. La alteración de la molécula de ADN original constituye una mutación que puede manifestarse en el fenotipo y hacer al individuo diferente del resto de sus congéneres. Es muy poco probable que cambiar al azar un trozo de información que la naturaleza ha refinado cuidadosamente a lo largo de millones de años resulte en algo bueno. Por lo general las mutaciones son desfavorables, incluso letales, para el organismo mutante. Pero ocasionalmente pueden no serlo y conferirle a dicho organismo alguna ventaja que le permita sobrevivir más fácilmente en su medio. Esta característica será transmitida a sus descendientes y un pequeño paso evolutivo se habrá dado.

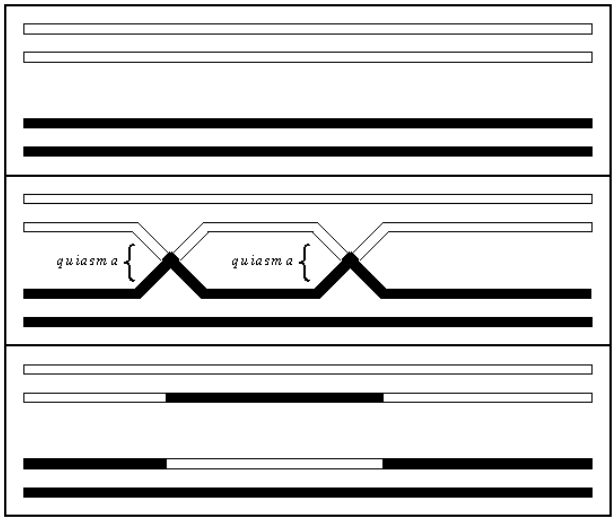


FIGURA F.I.2.B: *El proceso de cruzamiento de dos cromosomas. Al principio existen dos pares de cromosomas, un par del padre y otro de la madre; entre ellos se forman puntos de ruptura y unión (quiasmas) que definen segmentos llamados cromátidas. Se unen cromátidas complementarios para formar dos cromosomas híbridos.*

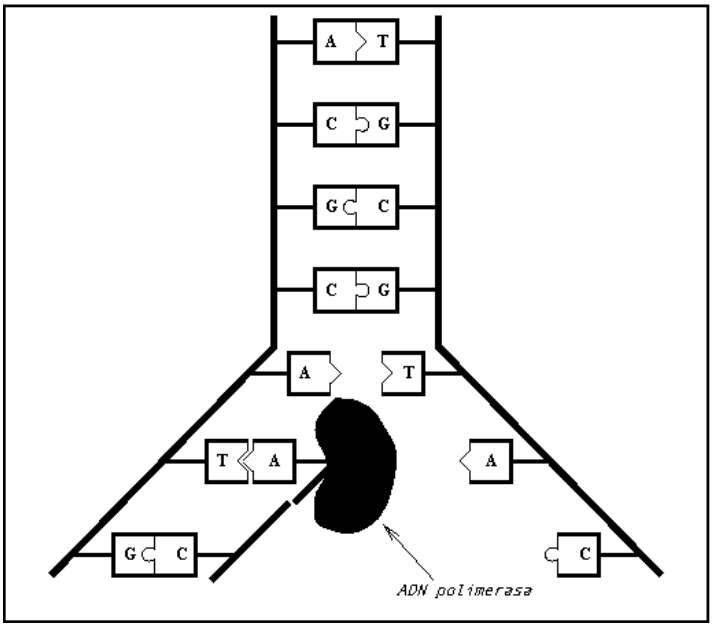


FIGURA F.I.2.C: *Proceso de replicación de la molécula de ADN. La molécula se abre como una cremallera. Una enzima (ADN-polimerasa) se encarga de colocar en cada fila los nucleótidos que corresponden. De cada fila se obtiene una nueva molécula de ADN.*

I.3 ALGORITMOS GENÉTICOS

Se mencionó ya que los algoritmos genéticos simulan el proceso de evolución natural. En esta sección se procurará aclarar la manera como se lleva a cabo esta simulación.

I.3.1 Codificación del dominio

En la naturaleza las características de los seres vivos, incluso aquéllas que los hacen óptimos para habitar en su medio, están determinadas por las proteínas que producen. A su vez, como hemos visto, estas proteínas (o más bien, los aminoácidos que las forman) se codifican en el material genético contenido en cada una de las células del individuo. Así pues, la naturaleza ha mapeado cada posible solución al problema de crear un individuo óptimo en una secuencia particular de bases que producirá ciertas proteínas, ha *codificado* el dominio del problema (todos los posibles individuos) mapeándolo al conjunto de todas las posibles secuencias de nucleótidos.

Así, para un algoritmo genético lo primero que se requiere es determinar en qué espacio se encuentran las posibles soluciones al problema que se pretende resolver. En caso de tener un problema de optimización de una función cuyo dominio es un subconjunto de los números reales, entonces este subconjunto es al que nos referimos. Pero el algoritmo opera sobre "códigos genéticos", sobre genotipos que se deberán mapear al espacio de soluciones. Es decir, es necesario *codificar* de alguna manera el dominio del problema para obtener estructuras manejables que puedan ser manipuladas por el AG. Cada una de estas estructuras constituye el equivalente al genotipo de un individuo en términos biológicos. El elemento del dominio del problema al que se mapea este genotipo es el análogo al fenotipo. Es frecuente que el código de los elementos del dominio del problema utilice un alfabeto binario (0's y 1's).

Una vez que se ha definido la manera de codificar los elementos del dominio del problema y se conoce la forma de pasar de un elemento a su código y viceversa, es necesario fijar un punto de partida. Los algoritmos genéticos manipulan conjuntos de códigos en generaciones sucesivas. Nuevamente haciendo una analogía, manipulan poblaciones de códigos. En éstas un código puede aparecer más de una vez. El algoritmo se encargará de favorecer la aparición en la población de códigos que correspondan a elementos del dominio que estén próximos a resolver el problema. En resumen, el algoritmo recibirá como entrada una población de códigos y a partir de ésta generará nuevas poblaciones, donde algunos códigos desaparecerán mientras que otros, que se mapean en mejores soluciones posibles, aparecen con más frecuencia hasta que se encuentra una satisfactoria o hasta que se cumple alguna otra condición de terminación. A lo largo del texto, los códigos en una población, es decir, los elementos de ésta serán llamados *individuos* y a los códigos en general, ya no en el contexto exclusivo de una población, se les denominará indistintamente *cromosomas*, *genotipo*, *genoma* o *código genético*, por analogía con los términos biológicos de donde surgen.

I.3.2 Evaluación de la población

En la naturaleza hay individuos más hábiles que otros para sobrevivir. En una manada de gacelas hay unas más rápidas que otras, hay algunas enfermas o propensas a enfermar, hay algunas más débiles que otras. Todas las características mencionadas señalan alguna diferencia entre los individuos. Además, esta diferencia es *relativa*, es decir, siempre está referida al resto de la población de gacelas. Se dice: "ésta es más rápida que el resto de la población" o "aquélla es más saludable que el promedio de sus congéneres". De alguna manera, siempre se relaciona al individuo con la población a la que pertenece. Si se considera cada una de estas características como medidas del desempeño de cada individuo, se está hablando de que el desempeño de cada individuo de la población está en función del desempeño de sus congéneres. Por ejemplo, ¿qué tan veloz debe ser

una gacela para evitar ser cazada por un cheetah? Si es más rápida que el cheetah está salvada. Pero lograr eso es difícil, casi no habría gacelas. Más bien la respuesta correcta es relacionar al individuo con el resto de su población y decir: debe ser más rápida que la gacela más lenta, de este modo el depredador perseguirá a otra (claro que después de la primera cacería la medida de desempe-ño cambia porque lo ha hecho la población misma). Al igual que en la naturaleza, en los algoritmos genéticos es necesario establecer algún criterio que permita decidir cuáles de las soluciones propuestas en una población son mejores respecto del resto de las propuestas y cuáles no lo son. Es necesario establecer, para cada individuo, una medida de desempeño relativa a la población a la que pertenece.

Para determinar cuáles de estos individuos corresponden a buenas propuestas de solución y cuáles no, es necesario calificarlos de alguna manera. Cada individuo de cada generación de un algoritmo genético recibe una *calificación* o, para usar el término biológico, una medida de su *grado de adaptación* (*fitness*). Éste es un número real no negativo tanto más grande cuanto mejor sea la solución propuesta por dicho individuo. El objetivo de este número es que permita distinguir propuestas de solución buenas de aquéllas que no lo son. Si el problema a resolver consiste en maximizar una función, entonces la calificación asignada a un individuo determinado debe indicar qué tan alto es el valor de la función en el elemento de su dominio codificado por el individuo. Si, en cambio, el problema es determinar la ruta más corta entre dos puntos, la calificación deberá ser tanto más alta cuanto más corto sea el camino codificado en el individuo que esté siendo calificado.

Evidentemente, al hablar de que a cada individuo de la población se le asigna una y sólo una calificación, se está hablando de una función que se denomina *función de adaptación*, cuya evaluación puede no ser sencilla y es, de hecho, lo que en la mayoría de los casos consume más tiempo en la ejecución de un algoritmo genético. Hay que tener en cuenta que se evalúa una vez en cada individuo de cada generación. Si un AG es ejecutado con una población de tamaño 100 durante 100 generaciones, la función es evaluada 10,000 veces. Además, puede darse el caso de que la función de evaluación no tenga una regla de correspondencia explícita, esto es, una expresión algebraica, y puede ocurrir incluso que la función cambie de generación en generación.

I.3.3 Selección

Una vez calificados todos los individuos de una generación, el algoritmo debe, al igual que lo hacen la naturaleza y el hombre, seleccionar a los individuos más calificados, mejor adaptados al medio, para que tengan mayor oportunidad de reproducción. De esta forma se incrementa la probabilidad de tener individuos "buenos" (con alta calificación) en el futuro. Si de una determinada generación

de individuos se seleccionaran sólo aquellos con una calificación mayor o igual que cierto número *c* para pasarlos a la siguiente generación, es claro que en ésta la calificación promedio superará *c* y por tanto al promedio de la generación anterior. La selección ocasiona que haya más individuos buenos, explota el conocimiento que se ha obtenido hasta el momento, procurando elegir lo mejor que se haya encontrado, elevando así el nivel de adaptación de toda la población. Más adelante en el texto se verá qué tan importante es esta explotación.

En principio podría parecer que es conveniente tener una estrategia de selección estricta para que mejore rápidamente la población y converja el algoritmo, es decir, que la población se acumule alrededor de un genotipo óptimo. Esto no es cierto. Lo que ocurrirá es que la población se acumulará rápidamente alrededor de algún individuo que sea bueno, comparativamente con el resto de los individuos considerados a lo largo de la ejecución del algoritmo, pero este individuo puede no ser el mejor posible. A esto se le suele llamar *convergencia prematura*. No se puede asegurar pero sí procurar que lo anterior no ocurra. Además de la explotación es necesario que exista exploración. El AG debe, no sólo seleccionar de entre lo mejor que ha encontrado, sino procurar encontrar mejores individuos. A esto se dedican los operadores que serán descritos a continuación, los que aseguran que en todo momento exista cierto grado de variedad en la población, procurando con ello que no se "vicie".

En la estrategia de selección normalmente se incluye un elemento extra que sirve de "ancla". Si sólo se hace selección forzando que sea más probable elegir al mejor individuo de la población pero sin asegurarlo, es posible que este individuo se pierda y no forme parte de la siguiente generación. Para evitar lo anterior se fuerza la selección de los mejores *n* individuos de la generación para pasar intactos a la siguiente. A esta estrategia se le denomina *elitismo* y puede ser generalizada especificando que permanezcan en la población los *n* mejores individuos de las pasadas *k* generaciones.

I.3.4 Cruzamiento

Durante la meiosis ocurre el proceso de producción de gametos. El código genético de los padres de un individuo se mezcla para producir gametos cuyo contenido genético es híbrido, es decir, una mezcla. De esta manera es posible que un individuo herede a sus descendientes las características mezcladas de sus propios padres, por ejemplo: el color de ojos del padre y el de cabello de la madre o, para aprovechar el ejemplo mencionado antes, es posible que una gacela herede la velocidad de su abuelo paterno y la fuerza de su abuela paterna, la salud de su abuelo materno y la agudeza visual de su abuela materna. Si estas características le confirieron a sus ancestros una alta aptitud de sobrevivencia, entonces este individuo será, con alta probabilidad, un individuo exitoso en su manada. La cruza

de los códigos genéticos de individuos exitosos favorece la aparición de nuevos individuos que hereden de sus ancestros características deseables.

En el contexto de los algoritmos genéticos reproducirse significa que, dados dos individuos seleccionados en función de su grado de adaptación, éstos pasen a formar parte de la siguiente generación o, al menos, mezclen sus códigos genéticos para generar *hijos* que posean un código híbrido. Es decir, los códigos genéticos de los individuos se *cruzan*. Existen muchos mecanismos de cruzamiento. En esta sección sólo se presenta uno de ellos pero todos tienen por objeto que el código de un individuo *A* y el de uno *B*, previamente seleccionados, se mezclen, es decir, se fragmenten y recombinen para formar nuevos individuos con la esperanza de que éstos hereden de sus progenitores las características deseables. El mecanismo de cruzamiento más común es el llamado *cruzamiento de un punto*, el cual se describe con detalle más adelante. Por ahora considérese como una analogía directa del proceso de cruzamiento descrito en la sección anterior, a propósito de la generación de gametos, con la formación de un único quiasma entre los cromosomas (individuos). A este quiasma es al que se le denominará más adelante *punto de corte*.

I.3.5 Mutación

Algunas veces, muy pocas de hecho, la ADN-polimerasa (la enzima encargada de replicar el código genético), se equivoca y produce una mutación, una alteración accidental en el código genético de los seres vivos.

Ocasionalmente algunos elementos del código de ciertos individuos de un algoritmo genético se alteran a propósito. Éstos se seleccionan aleatoriamente en lo que constituye el símil de una *mutación*. El objetivo es generar nuevos individuos, que exploren regiones del dominio del problema que probablemente no se han visitado aún. Esta exploración no presupone conocimiento alguno, no es sesgada. Aleatoriamente se buscan nuevas soluciones posibles que quizá superen las encontradas hasta el momento. Esta es una de las características que hacen aplicables los algoritmos genéticos a gran variedad de problemas: no presuponer conocimiento previo acerca del problema a resolver ni de su dominio, no sólo en la mutación sino en el proceso total. De hecho, el problema a resolver sólo determina la función de evaluación y la manera de codificar las soluciones posibles (la semántica de los códigos genéticos de los individuos). El resto de los subprocesos que constituyen el algoritmo son independientes y universalmente aplicables.

I.3.6 El algoritmo genético simple (AGS)

En su trabajo [7], Holland propone una manera de seleccionar individuos y de cruzarlos. Actualmente existen muchas otras propuestas, pero las de Holland constituyen aún hoy la base de muchos desarrollos teóricos y prácticos en el

área. Goldberg [6] retomó el algoritmo de Holland y lo popularizó llamándolo *algoritmo genético simple* (AGS). En éste se considera que los códigos genéticos están en binario. Explicado con detalle, el proceso de un AGS es:

1. Decidir cómo codificar el dominio del problema.
2. Generar un conjunto aleatorio (población inicial) de *N* posibles soluciones codificadas al problema. A ésta se le llamará la población actual.
3. Calificar cada posible solución (individuo) de la población actual.
4. Seleccionar dos individuos de la población actual con una probabilidad proporcional a su calificación.
5. Lanzar una moneda al aire (con probabilidad p_c cae cara).
6. Si cayó cara mezclar los códigos de los dos individuos seleccionados para formar dos híbridos, a los que llamaremos *nuevos individuos*.
7. Si cayó cruz llamamos a los individuos seleccionados *nuevos individuos*.
8. Por cada bit de cada nuevo individuo lanzar otra moneda al aire (con probabilidad p_m cae cara).
9. Si cae cara cambiar el bit en turno por su complemento.
10. Si cae cruz el bit permanece inalterado.
11. Incluir a los dos nuevos individuos en una *nueva población*.
12. Si la nueva población tiene ya *N* individuos, llamarla *población actual* y regresar al paso 3, a menos que se cumpla alguna condición de terminación.
13. Si no, regresar al paso 4.

En el algoritmo se utiliza el término "lanzar una moneda al aire" para decir un experimento de Bernoulli (aquel en el que pueden ocurrir exclusivamente dos eventos posibles, uno con probabilidad *p* y otro con probabilidad *1-p*). Es decir, el lanzamiento de una moneda "extraña" en la que no necesariamente ambas caras son equiprobables.

La condición de terminación, a la que se hace referencia en el paso 12, puede definirse de muchas maneras. Se puede fijar un número máximo de generaciones que se pretende ejecutar el algoritmo, o puede decidirse hacer alto cuando la mayoría de la población, digamos el 85%, tenga una calificación que esté dentro de 0.6 desviaciones estándar de la media. En fin, opciones hay muchas. Generalmente depende del problema o de las preferencias personales la decisión acerca de cuándo es conveniente detenerse.

En el paso 4 se menciona que hay que seleccionar dos individuos con probabilidad proporcional a su calificación. Este tipo de selección proporcional es también llamado de "ruleta" (*roulette wheel selection*) por lo siguiente: supóngase que se suman las calificaciones de todos los individuos de la población y esta suma es considerada el 100% de una circunferencia. Luego, a cada individuo se le asigna el trozo que le corresponde de ésta según su aportación a la suma de las calificaciones. Es decir, si la calificación de un individuo es x_i entonces le corresponde un segmento de circunferencia dado por la simple regla de tres:

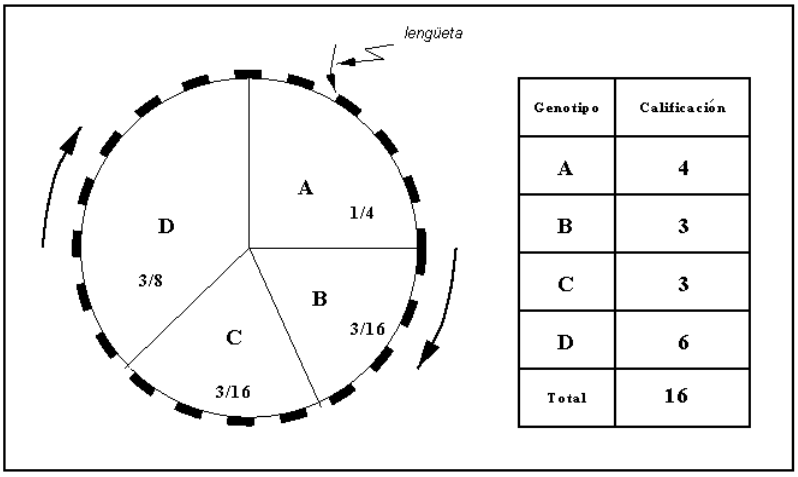
$$s = 2\pi \frac{x_i}{\sum_j x_j}$$

¿Qué ocurrirá entonces si se considera ésta cómo una ruleta y se coloca una lengüeta que roce el borde de ella? (figura F.I.3.A). La probabilidad de que dicha lengüeta quede en el arco correspondiente al individuo de calificación x_i, cuando la rueda se detenga tras realizar algunos giros, es:

$$p(i) = \frac{x_i}{\sum_j x_j}$$

lo que es proporcional a la calificación (x_i) del individuo.

Aún queda por aclarar cómo es que se mezclan los códigos de dos individuos para formar dos híbridos. En general hay muchas maneras de hacerlo. Al lector mismo se le pueden ocurrir muchas, sin embargo, la que se usa en el AGS y una de las más populares, es el cruzamiento de un punto (*1-point crossover*) ya mencionado. En este tipo de cruzamiento, dados dos individuos se elige aleatoriamente un punto de corte entre dos bits cualesquiera del cromosoma. Esto define segmentos izquierdos y derechos en cada genotipo. Se procede entonces a intercambiar los segmentos derechos (o los izquierdos, indistintamente) de cada individuo. De esto resultan, al igual que en el caso del cruzamiento de cromosomas, dos híbridos (figura F.I.3.B).



Genotipo	Calificación
A	4
B	3
C	3
D	6
Total	16

FIGURA F.I.3.A: *Selección proporcional o por ruleta*.

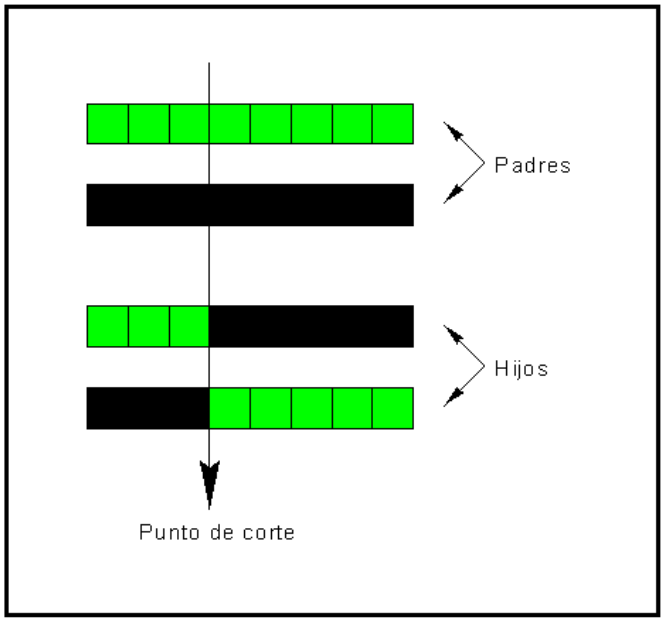


FIGURA F.I.3.B: *Cruzamiento de un punto (1-point crossover).*

A lo largo del texto se hará referencia frecuentemente al algoritmo genético simple como aquel caracterizado por tener:

1. Tamaño de población fijo en todas las generaciones.
2. Selección proporcional (de ruleta).
3. Cruzamiento de un punto. La probabilidad de cruza se mantiene fija para todas las generaciones y todas las parejas.
4. Mutación uniforme (todas las posiciones de las cadenas genéticas tienen la misma probabilidad de ser cambiadas). La probabilidad de mutación permanece fija para todas las generaciones y todas las posiciones de los individuos.
5. Selección no elitista. Esto es, no se copian individuos de una generación a otra sin pasar por el proceso de selección aleatoria (en este caso, proporcional).

I.3.7 Un ejemplo

Supóngase que se pretende encontrar el punto del intervalo [0, 0.875] donde la función:

$$f(x) = \left[1 - \left(\frac{11}{2}x - \frac{7}{2}\right)^2\right] \left[\cos\left(\frac{11}{2}x - \frac{7}{2}\right) + 1\right] + 2 \qquad \text{(I.3.A)}$$

tiene su máximo, usando para ello un algoritmo genético simple con probabilidad de cruza p_c = 0.70 y probabilidad de mutación p_m = 0.0666... = 1/15. En la figura F.I.3.C se muestra la gráfica de la función.

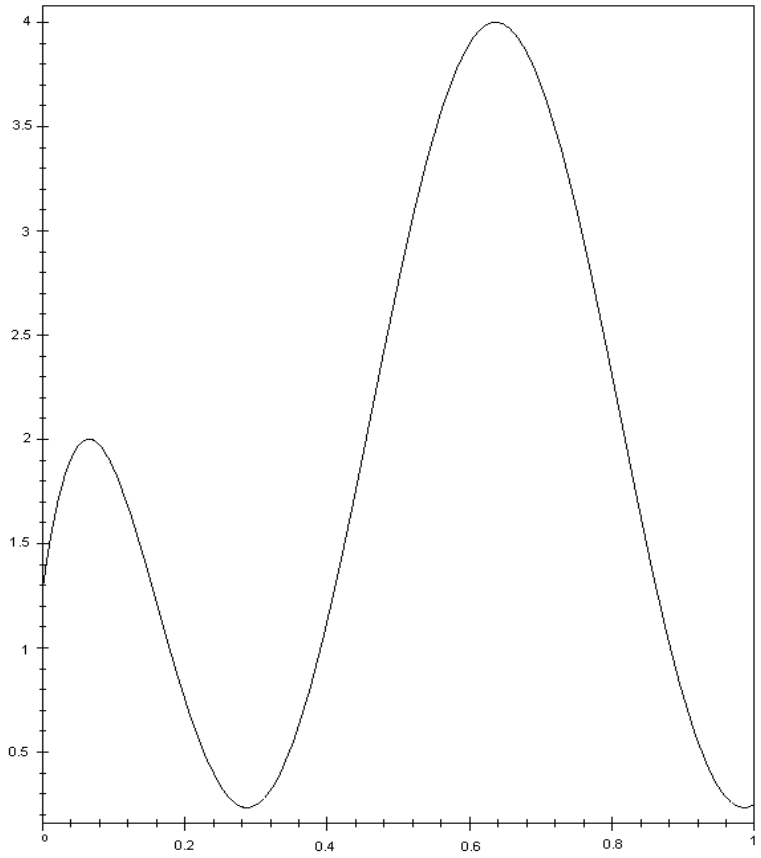


FIGURA F.I.3.C: *Gráfica de* f(x) *en el intervalo [0, 1]. Obsérvese que la función tiene dos puntos altos, uno de ellos es el máximo absoluto en el intervalo mientras el otro es sólo un máximo local.*

El primer paso del AGS es codificar el dominio de la función, que en nuestro caso es el intervalo [0, 0.875]. Hay que tomar en cuenta que siempre estos códigos serán almacenados en la memoria de la computadora donde se ejecutará el

algoritmo genético. Se pueden pensar muchas posibles maneras de codificar los elementos del dominio. Es posible, por ejemplo, colocar cada uno de los primeros *k* dígitos decimales del número (recuérdese que el dominio está constituido por números entre cero y 0.875) fijando el valor de *k* arbitrariamente; se puede pensar también en guardar estos números simplemente de la forma en que son representados los números reales por los compiladores de lenguajes de alto nivel, es decir, en el formato de punto flotante. No importa cuál mecanismo de codificación se utilice siempre se tendrá, por supuesto, un número finito de códigos disponibles para representar un conjunto infinito y no numerable de elementos del dominio, así que hay siempre una infinidad de elementos del dominio que no son representados. Con el fin de simplificar el ejemplo se codificarán los elementos del dominio en binario, usando 3 bits en la representación. Evidentemente se tienen entonces sólo 2^3=8 números representados (la codificación se muestra en la tabla T.I.3.A), y se puede pensar en que cada número binario de tres bits es, en realidad, la expansión en notación binaria pesada de la parte deci-mal de un número. Así por ejemplo, 101 representa $1\times2^{-1}+0\times2^{-2}+1\times2^{-3}$ = 1/2+ 1/8 = 0.625.

TABLA T.I.3.A: *Codificación y evaluación de los elementos del dominio de la función* f(x) *del ejemplo.*

No.	Código	E. Dominio *(x)*	*f(x)*
0	000	0.000	1.285
1	001	0.125	1.629
2	010	0.250	0.335
3	011	0.375	0.792
4	100	0.500	2.000
5	101	0.625	3.990
6	110	0.750	3.104
7	111	0.875	1.093

El máximo de la función en el intervalo se obtiene en:

$$x_m = \frac{7}{11} = 0.6363...$$

y el valor de la función en ese punto es justamente 4. Evidentemente x_m no está representado en nuestra codificación del dominio. El elemento más cercano representado es 0.625, así que se esperaría que el algoritmo genético encontrara la solución aproximada 101 que codifica al 0.625.

El siguiente paso es generar de manera aleatoria un conjunto de posibles soluciones. Nuevamente hay muchas formas de hacer esto: generar códigos válidos uniformemente distribuidos en el conjunto de posibles códigos, o generarlos con

una distribución normal o de Poisson, etc. Para que el proceso sea lo más imparcial posible, se ha elegido aquí una distribución uniforme (aunque la población es tan pequeña que esto no es notorio). Se ha seleccionado un tamaño de población de 5. En la tabla T.I.3.B se muestra la población inicial.

Ahora se procede a calificar a cada uno de los elementos de la población actual (en este caso la población inicial). En la tabla se muestra la calificación en la tercera columna (de izquierda a derecha). En el caso que nos ocupa se pretende maximizar una función. Ésta siempre toma valores no negativos en el intervalo que se codificó, así que un buen candidato para función de calificación, adaptación o *fitness* es, justamente, el valor mismo de la función. Es decir, la calificación de cada individuo será el valor de la función en el elemento del dominio que es codificado por dicho individuo.

Ahora se debe traducir esta calificación en una cierta medida de la oportunidad para reproducirse, es decir, de ser seleccionado para pasar a la siguiente generación o, al menos, pasar algunos descendientes. En la cuarta columna de la tabla T.I.3.B se muestra la probabilidad de selección de cada individuo de la población inicial. Para calcular la probabilidad de selección del *i*-ésimo individuo de la población basta dividir la calificación de *i* entre la suma total de las calificaciones de toda la población. Esta operación lleva implícita el mecanismo de selección uniforme o por ruleta que ya se mencionó. Pero no es necesario calcular esta probabilidad de selección.

TABLA T.I.3.B: *Población inicial P_0 generada uniformemente para el ejemplo.*

Índice	Individuos	Calificación	Prob. de selección	Calif. Acumulada
0	000	1.285	0.200	1.285
1	001	1.629	0.253	2.914
2	001	1.629	0.253	4.543
3	111	1.093	0.170	5.636
4	011	0.792	0.123	6.428
	Suma	*6.428*	*0.999*	
	Promedio	*1.286*		

Para efectuar la selección se hace lo siguiente:

1. Se genera un número aleatorio $r \in [0, 1]$.
2. Se multiplica r por la suma de las calificaciones la población (S), obteniéndose $c = r\,S$
3. Se establece la calificación acumulada (C_a) y el índice actual en cero: $C_a = 0,\ i = 0$

4. A la calificación acumulada se le suma la calificación del *i*-ésimo individuo: $C_a = C_a + calif(i)$
5. Si $C_a > c$ entonces el *i*-ésimo individuo es seleccionado
6. Si no, entonces se incrementa *i* y se regresa al paso 4

En nuestro caso, en la primera selección efectuada:

r = 0.213
c = 0.213 × 6.428 = 1.369

Dado que 1.369 está entre 1.285 y 2.914, calificaciones acumuladas para el individuo de índice 0 y el de índice 1, respectivamente, entonces el individuo seleccionado es precisamente el de índice 1.

En la segunda selección:

r = 0.314
c = 0.314 × 6.428 = 2.018

que nuevamente cae entre las calificaciones acumuladas para el individuo de índice 0 y el de índice 1, así que nuevamente es seleccionado el individuo de índice 1. Cabe enfatizar que la selección se hace *con reemplazo*, es decir, un mismo individuo puede ser seleccionado varias veces. Cada vez que se selecciona un individuo *no se quita de la población* de la que se está seleccionando, permanece en ella para que pueda ser elegido de nuevo.

Ahora que ya se poseen dos individuos seleccionados (que en realidad son el mismo), se procede a determinar si deben o no ser cruzados. Para ello:

1. Se genera un número aleatorio $d \in [0, 1]$.
2. Si $d < p_c$ entonces la pareja seleccionada (A, B) debe cruzarse.
3. Sea *l* la longitud de la cadena que constituye el código genético de cada individuo. Se genera un número aleatorio (*x*) entero entre 1 y *l*-1. Se indizan los bits de los individuos seleccionados de izquierda a derecha: 1, 2, ..., *l*.
4. Se pegan los bits de índices 1, ..., *x*-1 de A y los de índices *x*, ... *l* de B para formar un individuo híbrido llamado C y los bits 1, ..., *x*-1 de B se pegan con los de índices *x*, ..., *l* de A para formar un individuo híbrido D.
5. C y D son los nuevos individuos.
6. Si $d \geq p_c$ entonces la pareja seleccionada (A, B) no se debe cruzar. A y B son los *nuevos individuos.*

En el ejemplo que nos ocupa, dado que A y B son ambos iguales a 001, entonces no importa si se cruzan o no, los nuevos individuos serán nuevamente 001 ambos.

Pero en una repetición posterior del ciclo resultaron seleccionados el individuo 1 (001) y el individuo 3 (111), y en este caso al llevar a cabo el procedimiento descrito arriba:

d = 0.47

dado que d < 0.70 = p_c entonces 001 y 111 deben cruzarse
x = 1, así que A = 001 queda dividido en dos subcadenas: A_1 = 0, A_2 = 01 y B = 111 queda dividido en: B_1 = 1, B_2 = 11
Pegando A_1 y B_2 queda C = 011 y pegando B_1 y A_2 queda D = 101
C y D son los nuevos individuos

Cada vez que se tienen dos nuevos individuos se procede a ejecutar el procedimiento de mutación:

1. Se indizan nuevamente los bits de cada individuo de izquierda a derecha: 1, ..., l y para cada uno de los bits:
2. Se elige un número aleatorio q $\in$ [0, 1]
3. Si $q < p_m$ entonces el bit en turno se invierte (si valía 0 se transforma en 1 y viceversa)
4. Si $q \geq p_m$ entonces el bit en turno permanece sin cambio.

TABLA T.I.3.C: *Resultados del proceso genético aplicado a una cierta población inicial P_0 para resolver el problema de ejemplo. Los números en cursivas son el promedio de la columna.*

P_0	Calif.	P_1	Calif.	P_2	Calif.	P_3	Calif.
000	1.285	001	1.629	101	3.99	101	3.99
001	1.629	001	1.629	000	1.285	101	3.99
001	1.629	100	2	001	1.629	101	3.99
111	1.093	011	0.792	001	1.629	100	2
011	0.792	001	1.629	110	3.104	101	3.99
	1.285		*1.536*		*2.327*		*3.592*

En nuestro caso p_m = 1/15, es decir, en promedio sólo habrá un bit cambiado en cada generación. De hecho, en nuestro ejemplo el bit que resultó cambiado al obtener la primera generación fue el último bit de D = 101 que se acaba de obtener por cruzamiento, como resultado de esto se obtiene el 100 que aparece como individuo de índice 2 en la generación 1 mostrada en la tabla T.I.3.C. En esta tabla se muestra el proceso completo, generación a generación, y puede observarse cómo la calificación promedio de la población (números en cursivas al

final de cada columna de calificación), se va incrementando conforme se avanza en el tiempo. Si al principio se tenía una población dispersa con una calificación promedio de 1.285, al final se obtiene una población donde casi todos los individuos son 101 (el código más cercano al óptimo real) y la calificación promedio es de 3.592. De hecho, la población tiende a poseer sólo el individuo 101 (es el único punto óptimo) y solamente debido al operador de mutación hay algún individuo diferente (100 en el caso de P_3). Nótese también que, en las primeras etapas del proceso se tiende a reproducir con más frecuencia el individuo 001, donde la función tiene un máximo local. Esta situación se mantiene hasta que se encuentra el individuo 101, donde la función alcanza (aproximadamente) su máximo global.

I.4 PANORAMA DE OTROS MÉTODOS DE OPTIMIZACIÓN

En esta sección se presentan algunos métodos de optimización con el objetivo de que el lector tenga un panorama de las distintas alternativas que existen para el uso de los algoritmos genéticos. Algunos de los métodos presentados efectúan la búsqueda de puntos óptimos mediante estrategias que, en buena medida, dependen de eventos aleatorios. A este tipo de estrategias se les conoce como *heurísticas*. (Los algoritmos genéticos operan de esta manera.)

I.4.1 Métodos tradicionales

Las técnicas que aquí se muestran han sido seleccionadas porque son representativas de los dos grandes grupos de métodos de optimización: los directos (que no requieren más que los valores de la función objetivo), como simplex, y los de ascenso (que, además, requieren los valores de la o las derivadas de la función), como el de Newton y el de ascenso de máxima pendiente. Existen otros muchos métodos dentro de esta clasificación que son bastante más útiles que los que se muestran aquí. Sin embargo, el objetivo de esta sección es presentar, con propósitos didácticos, un panorama representativo de los métodos de optimización no heurísticos.

I.4.1.1 Método de Newton

En los cursos elementales de cálculo diferencial se les enseña a los alumnos que, bajo ciertas condiciones, es posible encontrar los puntos del dominio de una función $f:A \rightarrow B$ ($A, B \in \mathbb{R}$), donde ésta alcanza sus valores extremos (máximos y mínimos) encontrando las soluciones de la ecuación:

$$f'(x) = 0 \tag{I.4.A}$$

donde f' denota la derivada de f.

Resolver I.3.A analíticamente puede no ser fácil, así que en ocasiones es necesario encontrar soluciones numéricas aproximadas. Antes que cualquier otra cosa se especificarán las condiciones necesarias para que funcione el método que aquí se describe.

Se supondrá que: f es continua y derivable en su dominio al igual que su derivada f', la doble derivada f'' es distinta de cero en todo su dominio y existen ciertos puntos donde f' tiene un valor negativo y otros donde tiene un valor positivo.

Sean a y b dos puntos del dominio de f tales que $a < b$ y $f'(a)$ y $f'(b)$ tienen signos opuestos, como f' es continua entonces, por el teorema del valor intermedio, existe una η entre a y b tal que

$$f'(\eta) = 0.$$

Sean: $x_0=b$,l la recta tangente en $(x_0,f'(x_0))$ a la curva descrita por f', x_1 el punto donde l corta el eje x y θ el ángulo en que lo corta (figura F.I.4.A). Es evidente que:

$$x_1 = x_0 - (x_0 - x_1)$$

además:

$$\frac{f'(x_0)}{x_o - x_1} = tan(\theta) = f''(x_o)$$

por lo tanto:

$$x_0 - x_1 = \frac{f'(x_0)}{f''(x_0)}$$

es decir:

$$x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

En general, iterando el proceso:

$$x_r = x_{r-1} - \frac{f'(x_{r-1})}{f''(x_{r-1})} \tag{I.4.B}$$

Los valores que se obtienen de I.4.B conforme crece r son cada vez más próximos a algún valor $\xi \in A$, donde $f'(\xi)=0$. Si la función f' tiene varios puntos donde se anula, entonces para encontrar los demás será necesario probar con otros valores para x_0. El lector interesado en profundizar en este método puede consultar [1] o bien algún texto de análisis numérico.

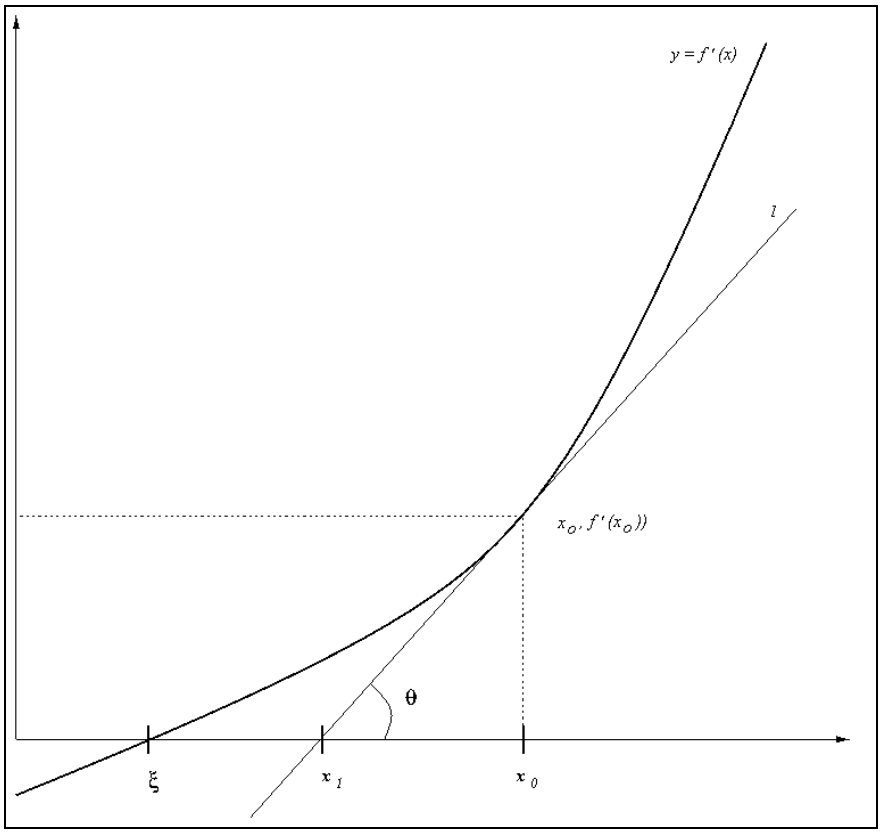


FIGURA F.I.4.A: *Ilustración del método de Newton para hallar los puntos donde* f' *se anula.* x_0 *es el punto inicial, a partir de éste se encuentra* x_1.

I.4.1.2 Búsqueda de Fibonacci

Este método sirve para buscar el máximo o mínimo de funciones de $\mathbb{R}\rightarrow\mathbb{R}$. El objetivo es acotar el punto donde la función alcanza su valor extremo por intervalos cada vez más pequeños, de esta forma al final se obtiene un intervalo (x, $x+\varepsilon$) en cuyo interior está el punto buscado.

Supóngase que se quiere localizar el punto donde una función alcanza su máximo y que es posible hacer hasta n evaluaciones de la función. La pregunta es: ¿De qué manera es posible determinar el valor de los puntos de muestra de la función, de tal forma que al final se tenga el intervalo de incertidumbre más pequeño posible? Supóngase ahora que se poseen tres puntos donde es conocido el valor de la función $x_1 < x_2 < x_3$ y sean: $L=x_2-x_1$ y $R=x_3-x_2$ donde L>R, tal y como se muestra en la figura F.I.4.B. Se desea determinar el punto x_4 tal que se

obtenga el intervalo de incertidumbre más pequeño posible. Si x_4 es elegido en (x_1, x_2) entonces:

Si $f(x_4) > f(x_2)$ el nuevo intervalo de incertidumbre será (x_1, x_2) de longitud x_2-x_1=L.
Si $f(x_4) < f(x_2)$ el nuevo intervalo de incertidumbre será (x_4, x_3) de longitud x_3-x_4.

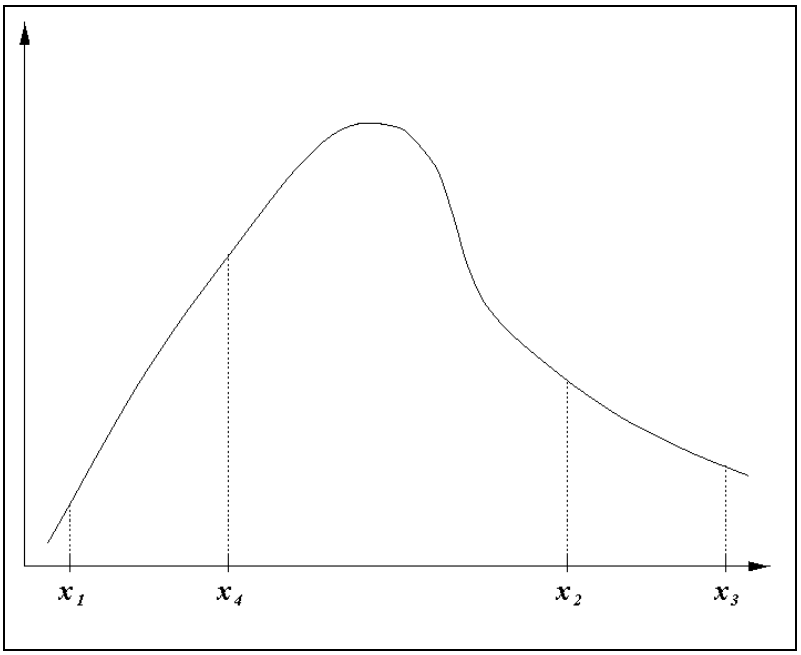


FIGURA F.I.4.B: *Ilustración del método de búsqueda de Fibonacci*.

Dado que no es posible determinar *a priori* cuál de estas dos cosas ocurrirá, es necesario elegir x_4 de forma tal que se minimicen al mismo tiempo x_2-x_1 y x_3-x_4. Esto se logra haciéndolas iguales, es decir, colocando x_4 en una posición simétrica con respecto a x_2.

En general, en cualquier nivel de iteración $n>3$ del algoritmo, si se denomina L_n la longitud del intervalo de incertidumbre, se tiene que: $L_{n-2} = L_{n-1} + L_n$. Además, si en ese nivel ocurre que x_n-x_{n-1} =ε entonces $L_{n-1} = 2L_n$-ε (véase figura F.I.4.C), de donde:

$$L_{n-1} = 2L_n-\varepsilon$$
$$L_{n-2} = L_{n-1} +L_n = 3L_n-\varepsilon$$
$$L_{n-3} = L_{n-2} +L_{n-1} = 5L_n-2\varepsilon$$

$$L_{n-4} = L_{n-3} + L_{n-2} = 8L_n - 3\varepsilon \qquad etc. \qquad \text{(I.4.C)}$$

La sucesión de números de Fibonacci es: $F_0=1$, $F_1=1$ y $F_k = F_{k-1} + F_{k-2}$ para $k=2,3,...$ Por lo que generalizando, es posible escribir:

$$L_{n-j} = F_{j+1}L_n - F_{j-1}\varepsilon \qquad j=1,2,...,n-1$$

Esta es la razón del nombre búsqueda de Fibonacci. El lector interesado puede consultar [1] o bien [12].

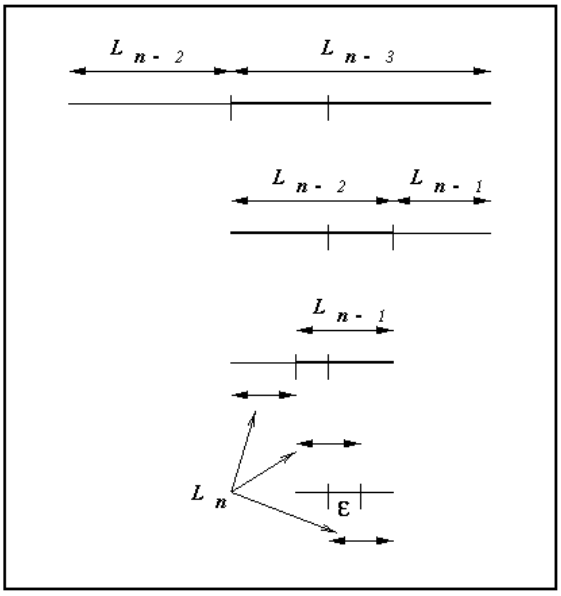


FIGURA F.I.4.C: *Relación entre las longitudes de los intervalos en la búsqueda de Fibonacci. Con líneas gruesas se han denotado los intervalos de incertidumbre.*

I.4.1.3 Ascenso de máxima pendiente (steepest ascent)

En los cursos de cálculo diferencial de varias variables se enseña a los alumnos que el gradiente de una función $f(\mathbf{x})$, que se denota con $\nabla f(\mathbf{x})$, es un vector que apunta en la dirección de mayor crecimiento de f.

Dado un punto $\mathbf{x}_0 \in \mathbb{R}^n$ del dominio de la función f: $\mathbb{R}^n \rightarrow \mathbb{R}$ si $\lambda_0 \in \mathbb{R}^+$ es un escalar positivo entonces en el vector: $\mathbf{x}_0 + \lambda_0 \nabla f(\mathbf{x}_0)$ el valor de f es mayor que en $\mathbf{x}_0$.

Aprovechando este hecho se define un proceso iterativo donde:

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + \lambda_{i-1} \nabla f(\mathbf{x}_{i-1}) \qquad \text{(I.4.D)}$$

En esta expresión λ_{i-1} maximiza:

$$\phi(\lambda) = f[\boldsymbol{x}_{i-1} + \lambda \nabla f(\mathbf{x}_{i-1})]$$

Este valor puede hallarse por procedimientos de búsqueda unidimensional como el de Fibonacci. En ocasiones esta "complicación" se omite y el valor de λ_{i-1} de la expresión I.4.D es determinado por procedimientos más empíricos. Por ejemplo, si en alguna iteración al utilizar un valor λ_i el resultado de la función en el nuevo punto no es mayor que en el actual, entonces se reduce el tamaño del "paso" que se debe dar desde allí.

Para aclarar un poco más este método se resolverá un ejemplo. Sea:

$$z = f(x,y) = -(x - 5)^2 - (y - 3)^2 \qquad \text{(I.4.E)}$$

se debe hallar el punto donde $f(x,y)$ alcanza su valor máximo (figura F.I.4.D). Arbitrariamente se elegirá como punto inicial $\boldsymbol{v}_0 = (1,1)$. De la expresión I.4.E se obtiene:

$$\nabla f(x,y) = (-2(x-5), -2(y-3))$$

de donde $\nabla f(1,1) = (8,4)$. Así que se debe elegir t_0 tal que se maximice:

$$\begin{aligned} f(t_0) &= f((1,1) + t_0(8,4)) \\ &= f(1 + 8t_0, 1 + 4t_0) \\ &= -(8t_0 - 4)^2 - (4t_0 - 2)^2 \\ &= -80t_0^2 + 80t_0 - 20 \end{aligned}$$

derivando esto e igualando a cero:

$$f'(t_0) = -80(2t_0 - 1) = 0$$

de donde:

$$t_0 = \frac{1}{2}$$

El nuevo punto sería, entonces: $\boldsymbol{v}_1 = (1,1)+(1/2)(8,4) = (5,3)$ y como $\nabla f(5,3) = 0$ se termina el proceso. El punto donde la función tiene su máximo es (5,3), el valor de la función en ese punto es 0. El lector interesado en profundizar puede consultar [9] ó [12].

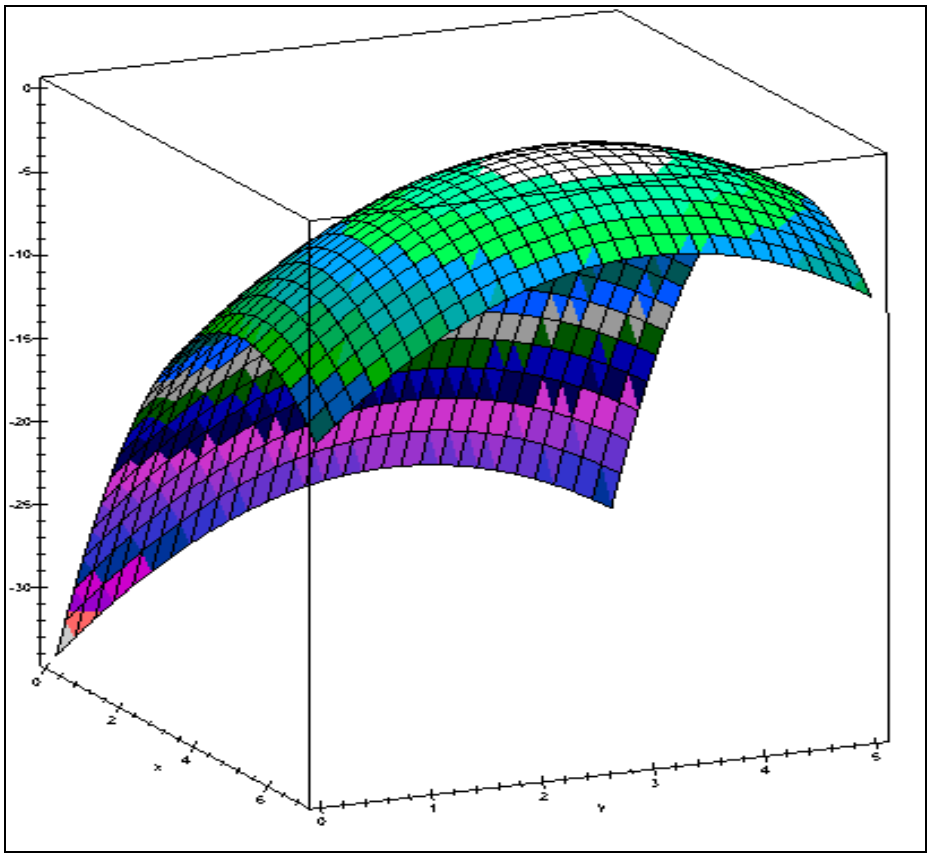


Figura F.I.4.D: *Gráfica de* $f(x,y) = -(x-5)^2-(y-3)^2$.

I.4.1.4 Simplex

Este es un método muy usual para resolver problemas de optimización de programación lineal. En éste se busca el punto donde una función lineal de varias variables (llamada función objetivo) tiene su máximo o mínimo cuando su dominio se ve restringido. Las variables deben tener cierto signo y sus valores deben satisfacer ciertas relaciones entre ellas, también lineales, expresadas como un conjunto de ecuaciones o desigualdades. Un ejemplo de problema de programación lineal es maximizar:

$$z = 2x_1 + 3x_2 \qquad \text{(I.4.F)}$$

$$\textit{restricciones} \begin{cases} x_1 + 2x_2 \le 6 \\ 2x_1 + x_2 \le 8 \end{cases}$$

$$\textit{restricciones de signo} \begin{cases} x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$$

En la figura F.I.4.E (cuatro páginas adelante) se muestra sombreado el dominio restringido de este problema. Dicho dominio se llama *región de factibilidad* y es donde se deben buscar los valores de las variables que maximizan I.4.F.

Antes de continuar se establecerán algunas definiciones.

- conjunto convexo: Un conjunto de puntos *S* es convexo si y sólo si cualquier segmento de resta que una a dos puntos de *S* está completamente contenido en *S*.
- punto extremo: En cualquier conjunto convexo *S*, un punto *p* de *S* es extremo si en cualquier segmento de recta completamente contenido en *S* que contenga a *p* éste es un extremo del segmento. En un círculo todos los puntos de la circunferencia son extremos y en un rectángulo sólo son los vértices.
- forma normal: Un problema de programación lineal está en forma normal si las restricciones aparecen expresadas como ecuaciones y se establece que todas las variables son mayores o iguales a cero. En nuestro caso el problema I.4.F en forma normal es:

$$\begin{aligned} z - 2x_1 - 3x_2 &= 0 \\ x_1 + 2x_2 + S_1 &= 6 \\ 2x_1 + x_2 + S_2 &= 8 \end{aligned}$$

$$\textit{restricciones de signo} \begin{cases} x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$$

Nótese que se han introducido nuevas variables para transformar las desigualdades en ecuaciones.

Un problema en forma normal puede ser expresado en términos matriciales. En nuestro caso el problema de ejemplo es:

$$A\boldsymbol{x} = \boldsymbol{b}$$

donde:

$$\mathrm{A} = \begin{pmatrix} 1 & -2 & -3 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 2 & 1 & 0 & 1 \end{pmatrix}$$

$$\mathbf{x} = \begin{pmatrix} z \\ x_1 \\ x_2 \\ S_1 \\ S_2 \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} 0 \\ 6 \\ 8 \end{pmatrix}$$

- solución básica: De un problema de *m* ecuaciones y *n* variables ($n \geq$ m) de la forma A$\boldsymbol{x}$ = $\boldsymbol{b}$ se obtiene cuando se hacen iguales a cero *n*-*m* variables (no básicas) y se resuelve el sistema para las restantes *m* variables (básicas). Evidentemente es posible obtener una solución básica si las columnas de A correspondientes a las *m* variables son linealmente independientes.
- solución factible básica: Es una solución básica donde todas las variables son no negativas.

Resulta que la región de factibilidad de cualquier problema de programación lineal es un conjunto convexo, y si el problema tiene solución, ésta será un punto extremo de la región de factibilidad. Además, hay una correspondencia uno a uno entre los puntos extremos del dominio de un problema de programación lineal y sus soluciones factibles básicas. Se dice que dos soluciones factibles básicas (sfb) son adyacentes si sus conjuntos de variables básicas (las que no fueron establecidas en cero) tienen *m*-1 variables en común (difieren en una variable, a lo más).

Simplex funciona aprovechando la correspondencia biyectiva que se mencionó. En términos generales el algoritmo es el siguiente:

1. Dado un problema de programación lineal, llevarlo a su forma normal y encontrar una sfb, a ésta llamarle la sfb actual.
2. Determinar si la sfb actual es óptima. Si lo es, terminar. Si no, encontrar una sfb adyacente a ella que mejore el resultado.
3. Regresar a 2 con la nueva sfb como la sfb actual.

Nótese que en un problema con *n* variables y *m* ecuaciones se tienen a lo más:

$$\binom{n}{m}$$

soluciones básicas y posiblemente no todas sean factibles (con todas las variables mayores o iguales a cero). Sin embargo, éste puede ser un número considerablemente grande. Según los resultados encontrados por Chvàtal [2], para re-

solver problemas de 50 variables con $m \leq 50$, simplex examina en promedio $2m$ soluciones básicas factibles antes de dar con la óptima.

Con más detalle, el algoritmo simplex opera como sigue:

1. Convertir el problema a su forma normal.
2. Obtener una sfb, llamarla la sfb actual.
3. Determinar si la sfb actual es óptima; si lo es, detenerse.
4. Si no, determinar cuál de las actuales variables no básicas se convertirá en básica y viceversa, para obtener un mejor valor en la función objetivo.
5. Usar reducción de Gauss-Jordan para obtener la nueva sfb y regresar a 3.

Falta por aclarar cómo ha de efectuarse el paso 4. La función objetivo, tal y como está expresada en la forma normal es:

$$z - a_1 x_1 - a_2 x_2 - \ldots - a_n x_n = 0 \qquad \text{(I.4.G)}$$

En un problema de maximización, ¿cuál variable contribuye más al valor de z? Evidentemente la que posee el coeficiente a_i más negativo en (I.4.G). Así que la variable que debe ingresar al conjunto de variables básicas en el paso 4 es aquella que en el renglón cero de A tenga el coeficiente más negativo, a ésta se le denomina variable entrante. Luego hay que calcular el cociente:

$$c_i = \frac{b_i}{a_{ij}}$$

para cada renglón i donde la variable entrante x_j posea un coeficiente positivo. Sea r el índice del renglón tal que:

$$c_r = min_i \{c_i\} = \frac{b_r}{a_{rj}}$$

En el algoritmo se elige $a_{r,j}$ como el pivote para efectuar la reducción de Gauss-Jordan de tal forma que la matriz resultante tendrá: $a'_{rj} = 1$ y $a'_{ij} = 0$ para toda $i \neq r$.

Este proceso quedará más claro resolviendo el ejemplo que se presentó desde el principio de la sección (expresión I.4.F). La matriz asociada con el problema, incluyendo una columna para los lados derechos de las ecuaciones es:

$$\mathrm{A} = \begin{pmatrix} 1 & -2 & -3 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 6 \\ 0 & 2 & 1 & 0 & 1 & 8 \end{pmatrix} \qquad \text{(I.4.H)}$$

El coeficiente más negativo del renglón cero es -3, que corresponde a x_2, así que ésta debe ser la variable de entrada. Todos los elementos de esa columna son positivos, c_1=6/2=3 y c_2=8/1=8, así que 3=$min\{c_1, c_2\}$. Es decir, el número 2 del segundo renglón, tercera columna, será el pivote que se debe convertir en 1. Efectuando las operaciones de reducción a I.4.H se obtiene:

$$A_1 = \begin{pmatrix} 1 & -\frac{1}{2} & 0 & \frac{3}{2} & 0 & 9 \\ 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 3 \\ 0 & \frac{3}{2} & 0 & -\frac{1}{2} & 1 & 5 \end{pmatrix}$$

Ahora es el -1/2 del renglón cero el que determina la variable de entrada (x_1) y el 3/2 de esa misma columna el que tiene el menor cociente (c_2=10/3) y por lo tanto el nuevo pivote. Aplicando nuevamente la reducción de Gauss-Jordan:

$$A_2 = \begin{pmatrix} 1 & 0 & 0 & \frac{4}{3} & \frac{1}{3} & \frac{32}{3} \\ 0 & 0 & 1 & \frac{4}{6} & -\frac{1}{3} & \frac{4}{3} \\ 0 & 1 & 0 & -\frac{1}{3} & \frac{2}{3} & \frac{10}{3} \end{pmatrix}$$

De donde se obtiene la solución ya que el máximo valor de z es 32/3 cuando x_1=10/3 y x_2=4/3, que, evidentemente, es una esquina (punto extremo) del dominio restringido del problema (véase figura F.I.4.E). El lector interesado en profundizar puede consultar [14].

I.4.2 Métodos heurísticos

I.4.2.1 Búsqueda tabú

Este método de búsqueda utiliza cierta memoria o conocimiento acerca del dominio para encontrar una buena solución al problema que se pretende resolver, o una solución, al menos, cercana a la óptima. El método se deriva del trabajo de Glover [5], y desde su creación ha recibido numerosas contribuciones y se han desarrollado algunas variantes.

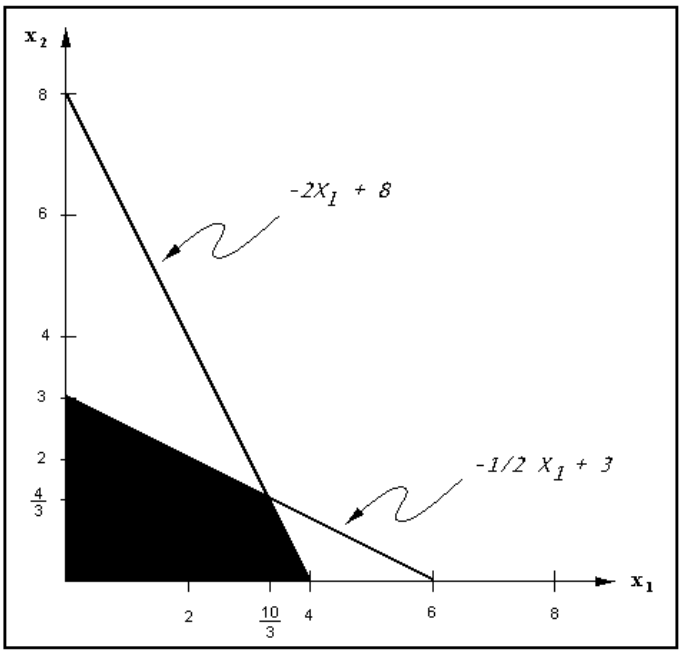


FIGURA F.I.4.E: *Dominio restringido del problema de programación lineal presentado como ejemplo.*

Sea X el dominio de búsqueda relacionado con algún problema y sea $x \in X$ una posible solución. Para buscar puntos mejores en X se define, de alguna manera conveniente, la vecindad de un punto cualquiera en X. De esta manera es posible buscar soluciones mejores que x entre sus vecinos, a los que se denotará con $N(x)$. Conforme pasa el tiempo y se va explorando más el dominio, será conveniente excluir algunos puntos que se sabe no son buenos (para no buscar en vano). De manera que la vecindad de un punto x cambia con el tiempo, depende de la historia (H) del proceso de búsqueda. Así que es conveniente denotar con $N(x)$ sólo a la vecindad inicial de x, y con $N(H,x)$ la vecindad de x como función de la historia. Algunos puntos que estaban en $N(X)$ y que, al paso del tiempo, resulten no ser buenos candidatos de solución, serán marcados como *tabú* y no podrán ser alcanzados desde x, es decir, no serán elementos de $N(H,x)$. También, conforme transcurre el tiempo la función que evalúa la conveniencia de cada solución posible se puede modificar y, en general, se denota con $b(H,x)$. Al principio de la ejecución del algoritmo es deseable identificar regiones donde se obtienen buenas soluciones y regiones donde se obtienen malas. Esto es posible si se posee suficiente diversidad, es decir, si se procura la exploración. En etapas tardías del algoritmo es mejor hacer una búsqueda más intensiva dentro

de las regiones buenas para aproximarse al óptimo. Esto es, en general, válido para varias heurísticas.

Cuando es costoso determinar qué elementos están en $N(H,x)$ se selecciona una muestra representativa de la vecindad $NC(H,x) \subseteq N(H,x)$.

En la búsqueda tabú la aleatoriedad es, si no ignorada, por lo menos minimizada. Es empleada de manera muy restringida en el entendimiento de que la búsqueda ha de ser un proceso sistemático [4]. Por supuesto, existe una variante llamada búsqueda tabú probabilística (*probabilistic tabu search*) en la que, a partir de un elemento x, se selecciona aleatoriamente el vecino que debe ser visitado a continuación.

En general el algoritmo de búsqueda tabú es:

PROCESS (**TABÚ**)
begin
  $x = selecc(X)$
  $x^* = x$
  $H = \varnothing$
  $max = b(H,x)$
  calcular ($NC(H,x)$)
  elegir x' *tal que* $b(H, x') = max \{b(H,y) \mid y \in NC(H,x)\}$
  repeat
    $x = x'$
    if ($b(H,x) > b(x^*)$)
    then
      $max = b(H,x)$
      $x^* = x$
    endif
    actualizar(H)
    actualizar($NC(H,x)$)
    elegir x' *tal que* $b(H, x') = max \{b(H,y) \mid y \in NC(H,x)\}$
  until (*cond. termina*)
  return (x^*)
end

El lector interesado en profundizar puede encontrar un análisis detallado de la búsqueda tabú en [4] y en [11].

I.4.2.2 Recocido simulado

Al igual que el método descrito en la subsección anterior, el recocido simulado (*simulated annealing*) es heurístico, y está basado en una analogía con el proceso físico seguido para obtener sólidos con configuraciones de energía mínima. Fue propuesto por primera vez por Metropolis en [10] y usado en optimización combinatoria por Kirkpatrick [8].

Probablemente el lector sepa que los sólidos cristalinos son los materiales más duros y que esto se debe a que su estructura molecular es muy simétrica, posee mínima energía. En Japón, desde hace milenios los maestros armeros elaboran los famosos sables de guerrero samurai siguiendo un proceso cuyos detalles se transmiten de generación en generación. Descrito superficialmente, este proceso consiste en calentar al blanco vivo una hoja de acero, doblarla sobre sí misma y golpearla con el martillo hasta que se integren las dos mitades y luego enfriarla sumergiéndola en agua. Estos pasos: calentar, doblar, golpear y enfriar se repiten decenas de veces y el resultado es una fina hoja de acero con el filo de una navaja de afeitar, pero con una dureza extraordinaria que la hace difícil de mellar. Esto es porque el acero de la hoja, a través de los múltiples calentamientos y enfriamientos, adquiere una configuración molecular cuasicristalina, con una muy baja energía.

Este mismo proceso es el que se sigue en la física de materia condensada para obtener sólidos con estados de mínima energía. Se calienta el sólido hasta hacerlo líquido y luego se le enfría muy lentamente para que las partículas de éste se acomoden en su estado de mínima energía o en alguno cercano a éste. Si el enfriamiento no se hace cuidadosamente es más probable que no se alcance el estado de energía mínima. En este caso hay que volver a calentarlo (en general menos que la vez anterior) y volver a enfriarlo con lentitud.

El recocido simulado se adapta naturalmente a problemas de minimización. Evidentemente, el problema puede modificarse para convertirlo en uno de maximización, pero aquí se supondrá que se pretende resolver problemas de minimización. En términos muy generales, la simulación del proceso de recocido en el algoritmo es:

1. Sea E_i la energía actual. Se desplaza un poco una molécula del estado presente. Con esto se cambia su posición respecto a las demás y se altera la energía del sistema que ahora será $E_{i+1} = E_i + \delta$. El valor de δ es calculado.
2. Si $\delta < 0$ entonces el nuevo estado se acepta y es el nuevo estado actual. Si $\delta \geq 0$ entonces el nuevo estado es aceptado con cierta probabilidad dada por:

$$e^{\frac{-\delta}{K_B T}}$$

   donde T es la temperatura y K_B una constante física, conocida como constante de Boltzmann.
3. Si conforme transcurre el tiempo se disminuye lentamente la temperatura, entonces el sólido alcanza un equilibrio térmico en cada temperatura y la probabilidad de estar en el estado caracterizado por tener energía E_i es:

$$\frac{1}{Z(T)} e^{\frac{-E_i}{K_B T}} \qquad \text{(I.4.I)}$$

donde $Z(T)$ es la función de partición definida como:

$$Z(T) = \sum_i e^{\frac{-E_i}{K_B T}}$$

De I.4.I es fácil ver que es mayor la probabilidad de tener un bajo nivel de energía (E_i pequeña, cercana a cero) y que esto es más difícil mientras menor sea la temperatura. De allí que sea importante disminuirla poco a poco.

El algoritmo descrito con más detalle es mostrado a continuación. En éste, X es el dominio de búsqueda, T_0 es la temperatura inicial, $N(i)$ es el conjunto de vecinos de i y $E(i)$ es la energía de la solución propuesta i.

```
PROCESS (RECOCIDO)
begin
 x = selecc(X)
 elegir T0
 T=T0
 repeat
  for (contador = 0) to L do
     elegir j ∈ N(i)
     δ = E(j)-E(i)
     if ((δ < 0) OR (aleatorio[0,1) < exp(-δ/T)))
     then
        i=j
     endif
  endfor
  reducir (T)
 until (cond. termina)
 return (i)
end
```

Nótese que, a diferencia de la búsqueda tabú, el estado actual del proceso en el recocido simulado depende sólo de su estado inmediato anterior. Esto ha hecho posible que se haga un análisis detallado de este método utilizando cadenas de Markov y, de hecho, ha sido probado teóricamente que el algoritmo converge asintóticamente a la solución global óptima. El lector interesado en profundizar puede encontrar un análisis detallado del recocido simulado en [3] y en [11].

I.4.3 Comparaciones

Los métodos de optimización presentados aquí son sólo una muestra de todos los existentes. El lector se preguntará, ¿cuál de ellos es el bueno? En optimización, al igual que en muchas otras situaciones de la vida cotidiana, no existe "el método". La conveniencia de cada uno depende del problema que se pretende resolver. Si éste involucra una función continua y derivable de una sola variable independiente, probablemente los métodos que se enseñan en los cursos de cálculo serán suficientes. En cambio, si el problema involucra una función muy complicada de decenas de variables independientes, de la que quizás no se conoce la regla de correspondencia, entonces muy probablemente (si no se tiene cierta predisposición al sufrimiento) se elija uno de los métodos heurísticos. Entre estos dos extremos hay una amplia variedad de problemas y métodos que se pueden utilizar. Hay mucho de donde escoger. La elección debe ir en proporción al problema. Coloquialmente, "no debe usarse un cañón para matar pulgas", pero tampoco se deben "cortar olmos con machete".

Hay, eso sí, métodos aplicables a casi cualquier problema, como los heurísticos, entre los que se encuentran los algoritmos genéticos y otros métodos que exigen mucho del problema, como los basados en el cálculo diferencial. En contraste, el grado de certeza que se puede tener en los resultados va en proporción inversa a la aplicabilidad. Casi nunca se puede estar completamente seguro de que el resultado entregado por un algoritmo heurístico sea el máximo o mínimo global o cuándo se debe detener el proceso de búsqueda.

En términos generales los métodos presentados aquí poseen las siguientes características:

- Newton: La función objetivo debe ser continua, dos veces derivable, con primera derivada continua y cuya segunda derivada no se anule.
- búsqueda de Fibonacci: La función objetivo debe ser continua y suave (sin saltos violentos).
- ascenso de máxima pendiente: La función objetivo debe ser continua y derivable. Calcular el gradiente puede no ser fácil.
- simplex: La función a optimizar debe ser lineal así como sus restricciones. Al ejecutarlo en computadora la exactitud del resultado depende de la estabilidad numérica de la matriz asociada.
- métodos heurísticos: El problema general en estos métodos es determinar *a priori* los valores de los parámetros de operación del algoritmo, por ejemplo: cómo establecer la dependencia entre la vecindad y la historia en búsqueda tabú, cómo ir modificando con el tiempo la temperatura en recocido simulado o qué valor asignarle a la probabilidad de mutación en un algoritmo genético.

La intención de esta sección ha sido presentar un panorama general de las alternativas a los algoritmos genéticos que existen en el área de la optimización, en el entendimiento de que no existe el mejor método de optimización en gene-

ral. Ante un problema se debe decidir, con base en sus características, cuál método será el más adecuado. No es válido tratar de resolver todos los problemas de una sola manera.

En el otro campo de aplicabilidad de los algoritmos genéticos, la inteligencia artificial, concretamente en el aprendizaje de máquina, ocurre algo similar. No todos los problemas se pueden resolver de la mejor manera con un sistema experto, con una red neuronal o con un algoritmo genético. Seguramente sería más fácil resolver algunos problemas si se utilizara alguna herramienta no usada hasta ahora, o mejor aún, una combinación de herramientas.

I.5 EJERCICIOS

1. Aproximadamente 4 de cada 100 personas posee un defecto en el brazo largo de su cromosoma 7. Este defecto genera el carácter recesivo que ocasiona la fibrosis quística. Por ser un carácter recesivo sólo se manifiesta cuando el genotipo que posee el individuo es homócigo (sus dos juegos de cromosomas poseen el defecto). Sin embargo, un individuo puede ser portador del defecto si al menos uno de sus cromosomas es defectuoso. Denótese con *C* un cromosoma normal y con *c* un cromosoma defectuoso. Entonces un individuo con genotipo *Cc* es portador pero no posee la enfermedad, uno con genotipo *CC* es completamente normal, no posee la enfermedad ni es portador del defecto, uno con genotipo *cc* posee la enfermedad y evidentemente, es portador. ¿Cuál es la probabilidad de que un par de padres portadores (Cc) del defecto engendren hijos con la enfermedad (cc)? ¿Cuál es la probabilidad de que engendren hijos portadores (Cc o cc)?

2. En el código genético cada terceto de bases codifica un aminoácido. En cada terceto se pueden repetir las bases. Calcule el número de posibles tercetos que se pueden formar con las cuatro bases (Adenina, Citosina, Guanina y Timina). Ahora bien, sólo hay veinte posibles aminoácidos con los que se construye cualquier proteína. ¿Qué significa esto?

3. El albinismo es un carácter recesivo, y se comporta igual que la enfermedad tratada en el ejercicio 1. Se tiene una población de ajolotes (*ambistoma mexicanum*) dividida en tres grupos. En el primero, todos los individuos son homócigos y de piel oscura, en el segundo todos los individuos son homócigos y albinos, en el tercer grupo todos los individuos son heterocigos y de piel oscura. Es posible describir simbólicamente cada grupo: G1=AA, G2=aa, G3=Aa. ¿Cuáles serán las proporciones de cada fenotipo (oscuro y albino) y de cada posible genotipo en los grupos obtenidos al hacer las siguientes cruzas? (G1,G1), (G1,G2), (G1,G3), (G2,G2), (G2,G3), (G3,G3).

4. Se tiene una población de un algoritmo genético constituida por individuos de seis diferentes genotipos $\{\alpha, \beta, \gamma, \delta, \varepsilon, \varphi\}$. Cada genotipo aparece con cierta frecuencia (F) en la población: $F(\alpha)=2$, $F(\beta)=3$, $F(\gamma)=2$, $F(\delta)=F(\varepsilon)=F(\varphi)=1$; la calificación f de cada representante de un genotipo es la misma: $f(\alpha)=5$, $f(\beta)=3$, $f(\gamma)=8$, $f(\delta)=17$, $f(\varepsilon)=22$, $f(\varphi)=9$. Si se utiliza selección proporcional, ¿cuál es la probabilidad de selección de cada genotipo? Sean $\Lambda^- = 0.25$ y $\Lambda^+ = 2 - \Lambda^- = 1.75$. Supóngase que la población del algoritmo genético se ordena por su calificación de menor a mayor (primero los peores, al último los mejores). Sea ahora:

$$P_i = \frac{1}{N}\left[\Lambda^- + \left(\Lambda^+ - \Lambda^-\right)\frac{i-1}{N-1}\right]$$

la probabilidad de seleccionar al i-ésimo individuo de la población en el orden mencionado (a este mecanismo de selección se le conoce en inglés como *linear ranking*), N es el tamaño de la población. Calcule las probabilidades de selección de cada individuo de la población. Compare lo obtenido con el caso de la selección proporcional.

5. Un problema NP-completo muy conocido es el del agente viajero: Se tiene un conjunto de ciudades etiquetadas de alguna manera $\{A_1, A_2, ..., A_n\}$. Cada ciudad está unida con todas las demás, es decir, cada ciudad es una vértice de una gráfica fuertemente conectada. Cada una de las aristas que une a una ciudad con otra tiene asociado cierto costo. Se elige una ciudad de partida A_p desde donde saldrá un agente viajero que debe regresar a A_p tras haber recorrido todas las demás ciudades, pasando por cada ciudad exactamente una vez. El problema consiste en encontrar el camino que minimice el costo total del viaje redondo. En el contexto de teoría de gráficas el problema es encontrar el ciclo Hamiltoniano de peso mínimo en una gráfica fuertemente conectada. Supóngase que se fija A_1 como el vértice inicial, cada posible solución se codificará como una cadena de n-1 símbolos de vértice (se excluye el vértice inicial que ya se sabe que es A_1), en cada una de estas posibles soluciones los n-1 símbolos son diferentes (no pueden repetirse vértices). ¿Cuál es la complejidad del problema para n vértices? Usando la codificación mencionada, ¿son útiles los mecanismos de selección, cruzamiento y mutación usados en el AGS?

6. Sean A=1001 y B=1010 dos genomas de una población de un algoritmo genético. ¿Cuáles posibles cadenas pueden ser generadas usando cruzamiento de un punto? Si ahora se seleccionan dos puntos de corte en los genomas y se intercambian los segmentos intermedios de cada genoma, ¿cuáles cadenas se pueden generar? Haga lo mismo con A=10011 y B=10100. ¿Puede

concluir algo acerca del número de cadenas alcanzables desde A y B en función del tipo de cruzamiento?

7. Sea A=001010110110 una cadena genética y sea $P_m = 0.25$ la probabilidad de mutación. ¿Cuál es el número esperado de bits de A que serán invertidos ($\mu(A)$)? Sea M(A) el conjunto de cadenas que difieren de A en, exactamente, $\mu(A)$ bits. ¿Cuántos elementos hay en M(A)? ¿De qué manera depende el tamaño de M(A) de la probabilidad de mutación?

8. Optimice las siguientes funciones por el método de Newton:

$$f(x) = (x-3)^2 + 2$$
$$g(x) = \frac{1}{1+x^2}$$

Analice el comportamiento en ambos casos.

9. Optimice la siguiente función por el método de ascenso de máxima pendiente:

$$f(x,y) = -\left(x - \frac{1}{2}\right)^2 - (y+4)^2$$

10. Utilice Simplex para maximizar:

$$z = 3x_1 + 2x_2$$

sujeta a las restricciones:

$$2x_1 + x_2 \leq 100$$
$$x_1 + x_2 \leq 80$$
$$x_1 \leq 40$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

I.6 PROGRAMACIÓN

1. Elabore un programa que implante un algoritmo genético simple. Maneje poblaciones de 100 individuos, probabilidad de cruza entre 40% y 90%, probabilidad de mutación entre 0.005 y 0.1, haga ejecuciones de hasta 200 generaciones. Pruebe maximizando las siguientes funciones en el intervalo [0, 1]:

$$f_1(x) = 1 - \left(x - \frac{1}{2} \right)^2$$
$$f_2(x) = 36 - \left((6x - 3)^2 - 3 \right)^2$$
$$f_3(x) = 1 - \left(x - \frac{1}{2} \right)^6$$
$$f_4(x) = \lfloor 10x \rfloor$$
$$f_5(x) = \text{sen}^2 (4\pi x)$$

Invente una función que posea su máximo en una vecindad muy pequeña y pruebe su programa con ella. Reporte sus resultados.

2. Elabore un programa que implante el mecanismo de recocido simulado o de búsqueda tabú y maximice las funciones del ejercicio anterior. Compare los resultados.

I.7 REFERENCIAS

[1] Bunday, B. D., *Basic Optimization Methods*, Edward Arnold Publishers, 1984.
[2] Chvàtal, V., *Linear Programming*, Freeman, 1983.
[3] Downsland, Kathryn, "Simulated Annealing", en *Modern Heuristic Techniques for Combinatorial Optimization Problems*, editado por Colin R. Reeves, John Wiley & Sons, 1993, pp. 20-69.
[4] Glover, F. y M. Laguna, *Tabu Search*, Kluwer Academic Publishing, 1997.
[5] Glover, F., "Future Paths for Integer Programming and Links to Artificial Intelligence", *Computers and Operations Research*, No. 5, 1986, pp. 553-549.
[6] Goldberg, D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989.
[7] Holland, J. H., *Adaptation in Natural and Artificial Systems*, 2a ed., MIT Press, 1992.
[8] Kirkpatrick, S., C. Gelatt y M. Vecchi, "Optimiztion by Simulated Annealing", *Science*, No. 220, 1983, pp. 671-679.
[9] Marsden, J., y A. Tromba, *Cálculo Vectorial*, Fondo Educativo Interamericano, 1981.
[10] Metropolis, N., A. Rosenbluth, A. Teller y E. Teller, "Equations of State Calculations by Fast Computing Machines", *The Journal of Chemical Physics*, Vol. 21, No. 6, 1995, pp. 1087-1092.

[11] Reeves, C. (editor), *Modern Heuristic Techniques for Combinatorial Problems*, John Wiley & Sons, 1993.
[12] Shoup, T., y Farrokh Mistree, *Optimization Methods with Applications for Personal Computers*, Prentice Hall, 1987.
[13] Smith-Keary, P., *Genetic, Structure and Function*, Macmillan Press, 1979.
[14] Winston, W. L., *Introduction to Mathematical Programming, Applications and Algorithms*, 2a ed., Duxbury Press, 1995.

II. FUNDAMENTOS MATEMÁTICOS

II.1 TERMINOLOGÍA

II.1.1 Función de adaptación

LOS algoritmos genéticos son mecanismos de búsqueda que operan sobre un conjunto de códigos posiblemente muy grande, pero finito. Del dominio de búsqueda se toman iteradamente conjuntos de muestras y cada elemento de una muestra es calificado dependiendo de qué tan bien cumpla con los requisitos de aquello que es buscado. Luego son seleccionados los elementos, a los que también llamaremos *individuos*, que cumplan mejor con dichos requisitos. Éstos se multiplican en las siguientes muestras y son sometidos a ciertos operadores que emulan a los que funcionan en la naturaleza.

Uno de los elementos esenciales involucrados en este proceso es la calificación asignada a cada individuo, a la que también llamaremos *grado de adaptación* a lo largo del texto. Esta calificación es un número mayor o igual a cero, y en conjunto con la selección se convierte en una medida de la posibilidad de reproducción para cada individuo. Esto es, existe una función de adaptación que asocia a cada individuo de la muestra (población) con un número real no negativo. Mientras más grande sea el valor asignado a un individuo dado mayor será la probabilidad de éste de ser seleccionado para formar parte de la siguiente generación de muestras.

II.1.2 Esquemas

Se ha creado una notación para los conjuntos de cadenas binarias (cromosomas) mediante cadenas compuestas de tres símbolos, a saber: {0,1,*}. Considérese un conjunto de cadenas que poseen valores comunes en ciertas posiciones. Para construir la cadena que denota este conjunto, basta colocar en las posiciones donde las cadenas coinciden el valor explícito que tienen, y en las posiciones donde los valores no coinciden se coloca un *. De esta manera por ejemplo, 1*0* denota el conjunto {1000,1001,1100,1101}. A las cadenas compuestas de 1, 0 y * se les denomina *esquemas*.

El número de posiciones con valor explícito de un esquema H se denomina *orden* del esquema y se denota por $o(H)$. La distancia entre la primera y la última posición explícita se denomina *longitud de definición* (*defining length*) y se denota como $\delta(H)$. En la figura F.II.1.A se ilustran esos conceptos.

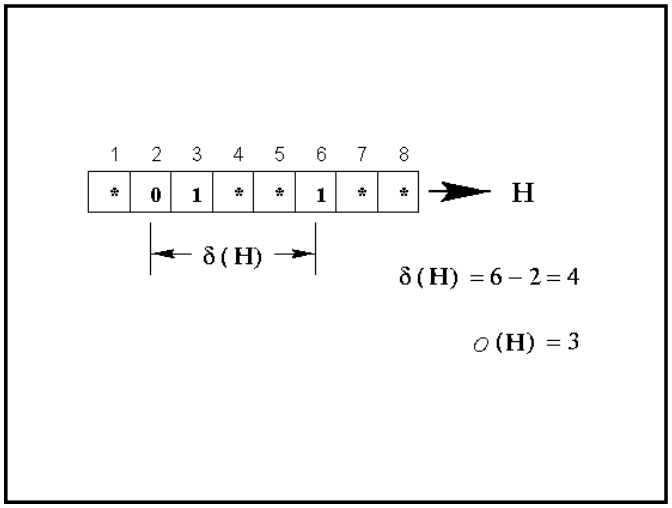


FIGURA F.II.1.A: *Orden y longitud de definición de un esquema.*

II.2 EL TEOREMA DEL ESQUEMA

Hasta la fecha, el modelo matemático más usual para describir el comportamiento de los algoritmos genéticos está basado en el *teorema del esquema*. Éste fue planteado originalmente por Holland [7] para el AGS, y se caracteriza por utilizar selección proporcional y cruzamiento de un punto, entre otras cosas. En esta sección se deducirá el teorema procurando ser un poco más general que en otras presentaciones (como las de [6] u [8]), pues se considerarán tipos de cruzamiento distintos del tradicional.

II.2.1 Selección

Supóngase que en la población de tamaño N de generación t de un algoritmo genético existen k representantes del esquema H y que f_1, f_2, ..., f_k son los valores de la función de adaptación para los k representantes. Si se selecciona aleatoriamente un miembro de la población usando el mecanismo de la ruleta, la probabilidad de que éste sea el i-ésimo representante del esquema H es:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

Entonces, la probabilidad de seleccionar algún representante del esquema *H* es:[1]

$$p_H = \frac{f_1}{\sum_{j=1}^{N} f_j} + \frac{f_2}{\sum_{j=1}^{N} f_j} + ... + \frac{f_k}{\sum_{j=1}^{N} f_j} = \frac{f_1 + f_2 + ... + f_k}{\sum_{j=1}^{N} f_j} \qquad \text{(II.2.A)}$$

Sean $\overline{f}(H)$ el valor de adaptación promedio de los representantes del esquema *H* y *m*(*H*,*t*) el número de representantes del esquema en la población de generación *t* (i.e., *m*(*H*,*t*)=*k*). Por definición:

$$\overline{f}(H) = \frac{f_1 + f_2 + ... + f_k}{m(H,t)}$$

de donde:

$$f_1 + f_2 + ... + f_k = \overline{f}(H) m(H,t)$$

sustituyendo en II.2.A:

$$p_H = m(H,t) \frac{\overline{f}(H)}{\sum_{j=1}^{N} f_j} \qquad \text{(II.2.B)}$$

esto denota la probabilidad de seleccionar un representante del esquema *H* en la población de generación *t*.

Sea $\overline{f}$ el valor promedio de adaptación de la población de generación *t*

$$\overline{f} = \frac{\sum_{j=1}^{N} f_j}{N}$$

entonces, si se seleccionan *N* individuos de la población de generación *t* para la generación *t+1*, el valor esperado de individuos seleccionados del esquema *H* es:

$$m(H,t+sel) = N\, m(H,t) \frac{\overline{f}(H)}{\sum_{j=1}^{N} f_j} = m(H,t) \frac{\overline{f}(H)}{\overline{f}}$$

Hay que reiterar que las expresiones presentadas son válidas sólo para el tipo de selección proporcional o de ruleta (no son ciertas en general).

[1] Suponiendo que cada selección es un evento independiente.

II.2.2 Cruzamiento

Supóngase ahora que se aplica algún tipo de cruzamiento con cierta probabilidad p_c sobre los individuos previamente seleccionados. Considérese lo que ocurriría con aquellas cadenas pertenecientes a un cierto esquema *H*. Algunas de estas cadenas se cruzarían con otras de forma tal que la cadena resultante ya no sería representante del esquema *H*, es decir, el esquema se rompería. Otras no serían seleccionadas para cruzarse y simplemente pasarían intactas a la siguiente generación y habría otras más que originalmente no eran representantes del esquema y que al cruzarse generarían cadenas de *H*. El valor esperado de cadenas representantes de *H* que han sido seleccionadas y a las que no se les aplica cruzamiento es:

$$(1-p_c)\, m(H,t)\, \frac{\overline{f}(H)}{\overline{f}} \qquad \text{(II.2.C)}$$

Sea p_r la probabilidad de ruptura del esquema *H* bajo el tipo de cruzamiento que esté siendo utilizado.[2] El valor esperado del número de cadenas representantes de *H* que fueron seleccionadas y permanecen en el esquema después de aplicárseles cruzamiento es:

$$p_c\left(m(H,t)\, \frac{\overline{f}(H)}{\overline{f}} (1-p_r) \right) \qquad \text{(II.2.D)}$$

A la expresión II.2.D habría que hacerle una pequeña corrección para considerar aquellas cadenas que originalmente estaban fuera del esquema y que después de cruzarlas generaron representantes de él. Sea *g* el número de cadenas ganadas por el esquema *H* mediante el mecanismo descrito, reescribiendo II.2.D se tiene:

$$p_c\left(m(H,t)\, \frac{\overline{f}(H)}{\overline{f}} (1-p_r) + g \right) \qquad \text{(II.2.E)}$$

Resumiendo II.2.C y II.2.E, el valor esperado del número de representantes del esquema *H* tras haber efectuado selección y cruzamiento es:

$$m(H,t+sel+cru) = (1-p_c) m(H,t)\, \frac{\overline{f}(H)}{\overline{f}} + p_c\left(m(H,t)\, \frac{\overline{f}(H)}{\overline{f}} (1-p_r) + g \right)$$

si excluimos *g*:

[2] En términos generales p_r depende del tipo de cruzamiento así como del orden del esquema $o(H)$ y de su longitud de definición δ(H).

$$m(H,t+sel+cru) \geq (1-p_c)m(H,t)\frac{\overline{f}(H)}{\overline{f}} + p_c\left(m(H,t)\frac{\overline{f}(H)}{\overline{f}}(1-p_r)\right)$$

factorizando:

$$m(H,t+sel+cru) \geq m(H,t)\frac{\overline{f}(H)}{\overline{f}}((1-p_c)+p_c(1-p_r))$$

es decir:

$$m(H,t+sel+cru) \geq m(H,t)\frac{\overline{f}(H)}{\overline{f}}(1-p_c p_r) \qquad \text{(II.2.F)}$$

II.2.3 Mutación

Por último, hay que considerar el efecto de las mutaciones. Supóngase que una mutación se aplica con probabilidad p_m y que tiene el efecto de invertir un bit (cambiar un 1 por un 0 ó viceversa). Para que una cadena representante del esquema *H* permanezca en él tras una mutación, debe ocurrir que ninguno de los bits definidos del esquema sea invertido. Recuérdese que el número de bits definidos del esquema es *o*(*H*). La probabilidad de que un bit no sea invertido por una mutación es 1-p_m, así que la probabilidad de que ninguno de los bits definidos sea invertido, suponiendo que el invertir cada bit es un evento independiente, es:

$$\mu(p_m) = (1\text{-}p_m)^{o(H)} \qquad \text{(II.2.G)}$$

Añadiendo II.2.G a la expresión II.2.F se tiene:

$$m(H,t+1) \geq m(H,t)\frac{\overline{f}(H)}{\overline{f}}(1-p_c p_r)(1-p_m)^{o(H)} \qquad \text{(II.2.H)}$$

A esta expresión se le conoce como el *teorema del esquema*, y existen diversas versiones de éste, todas ellas equivalentes. Goldberg, por ejemplo [6], prefiere desarrollar la expresión II.2.G en serie de MacLaurin (o bien mediante el teorema del binomio) para simplificar, dado que, en principio, p_m es un número muy cercano a cero:

$$\mu'(p_m) = \frac{d(1-p_m)^{o(H)}}{dp_m} = -o(H)(1-p_m)^{o(H)-1}$$

de donde:

$$\mu(0) = 1 \quad \text{y} \quad \mu'(0) = -o(H)$$

por lo tanto:

$$\mu(p_m) = (1\text{-}p_m)^{o(H)} \approx 1 - o(H)\, p_m \qquad \text{(II.2.I)}$$

Añadiendo II.2.I a II.2.F:

$$\begin{aligned} m(H,t+1) &\geq m(H,t)\frac{\overline{f}(H)}{\overline{f}}(1-p_c p_r)(1-o(H)p_m) \\ &= m(H,t)\frac{\overline{f}(H)}{\overline{f}}(1-p_c p_r - o(H)p_m + o(H)p_m p_c p_r) \\ &\geq m(H,t)\frac{\overline{f}(H)}{\overline{f}}(1-p_c p_r - o(H)p_m) \end{aligned}$$

finalmente Goldberg escribe:

$$m(H,t+1) \geq m(H,t)\frac{\overline{f}(H)}{\overline{f}}(1-p_c p_r - o(H)p_m) \qquad \text{(II.2.J)}$$

En otras versiones se encuentra todo dividido por el tamaño de la población, de tal forma que todo queda expresado en términos de probabilidad:

$$p(H,t+1) \geq p(H,t)\frac{\overline{f}(H)}{\overline{f}}(1-p_c p_r)(1-p_m)^{o(H)} \qquad \text{(II.2.K)}$$

o bien:

$$p(H,t+1) \geq p(H,t)\frac{\overline{f}(H)}{\overline{f}}(1-p_c p_r - o(H)p_m) \qquad \text{(II.2.L)}$$

II.2.4 Tipos de cruzamiento

Como se mencionó, existen diversos tipos de cruzamiento. En las expresiones II.2.H–II.2.L aparece una cierta probabilidad genérica p_r que depende del tipo de cruzamiento que se use y de las características del esquema que se analice. Los cruzamientos más utilizados son tres, a saber:

a) Cruzamiento de un punto. Se elige un punto de corte y se intercambian los segmentos análogos de las dos cadenas ([1], [6], [12]).
b) Cruzamiento de dos puntos. Se eligen dos puntos de corte y se intercambian los segmentos medios de ambas cadenas, se considera a los extremos de la cadena como sitios contiguos, i.e., como un anillo ([2], [12]).

c) Cruzamiento uniforme. Para cada posición de bit de una cadena a generar se elige aleatoriamente el bit de la misma posición de alguna de las cadenas generadoras ([2], [11], [12]).

Cada uno de estos tipos está ilustrado en la figura F.II.2.B. A continuación se determinará cuál debe ser la expresión para p_r en cada caso. En lo que sigue, l representará la longitud total de las cadenas de bits ($l > 1$). Recuérdese que $o(H)$ denota el número de bits explícitos (fijos) de un esquema y que $\delta(H)$ denota la distancia entre el primero y el último lugar fijos de un esquema.

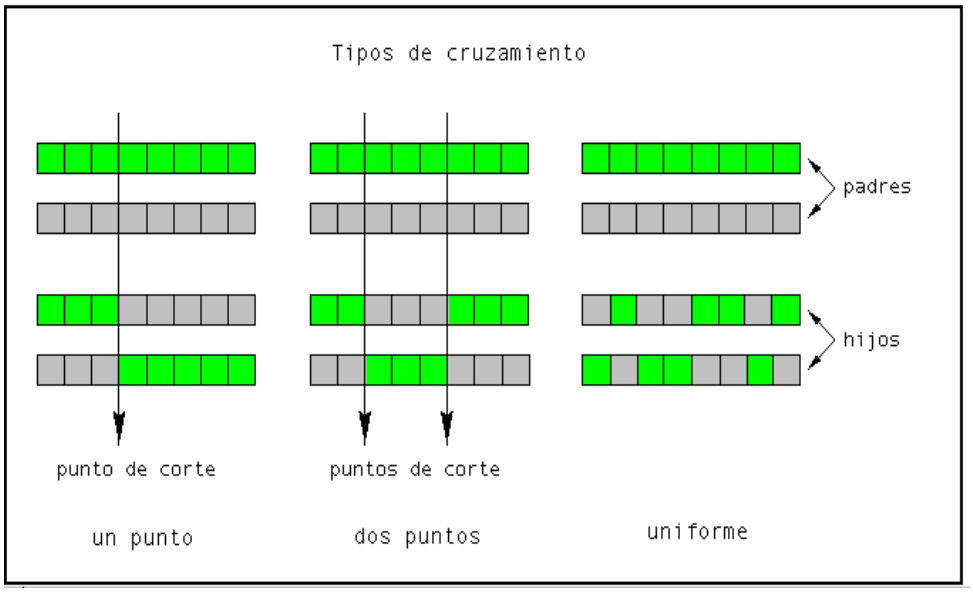


FIGURA F.II.2.B: *Tres tipos de cruzamiento.*

II.2.4.1 Cruzamiento de un punto

Si las cadenas se cortan en un solo punto, un esquema se romperá cuando la cadena que lo representa es cortada en algún punto entre los bits fijos del esquema, es decir, si se corta en algún lugar de los abarcados por $\delta(H)$.

En una cadena de longitud l existen l-1 posibles puntos de corte, así que la probabilidad de romper el esquema H con un corte es:

$$p_r \leq \frac{\delta(H)}{l-1}$$

El símbolo ≤ se debe a que puede ocurrir que al cruzarse dos instancias de un esquema se generen, nuevamente, instancias de dicho esquema.

II.2.4.2 Cruzamiento de dos puntos

Existen exactamente $\binom{l}{2}$ maneras de elegir dos puntos de corte de una cadena de longitud $l > 1$ considerándola como unida en sus extremos. Supóngase que en esta cadena se encuentra un esquema de longitud definida $\delta(H)$ y orden $o(H)$. Si el orden del esquema es dos ($o(H)$=2), entonces H se rompe cuando uno de los puntos de corte cae dentro del segmento abarcado por $\delta(H)$ y el otro cae fuera de ese segmento, en alguno de los l-$\delta(H)$ lugares restantes. Es decir, la probabilidad de ruptura de un esquema de orden dos es:

$$P_{\delta(H),2} = \frac{\delta(H)(l-\delta(H))}{\binom{1}{2}} = \frac{2\delta(H)(l-\delta(H))}{l(l-1)} \qquad \text{(II.2.M)}$$

En cambio, si se supone que el esquema tiene definidos todos los lugares abarcados por $\delta(H)$, es decir, $o(H)= \delta(H)+1$, entonces el esquema también se rompe siempre que los dos puntos de corte caen dentro del segmento considerado por $\delta(H)$, así que modificando II.2.M se obtiene:

$$P_{\delta(H),\delta(H)+1} = \frac{\delta(H)(l-\delta(H)) + \binom{\delta(H)}{2}}{\binom{1}{2}}$$

En general:

$$p_{\delta(H),2} \leq p_r \leq P_{\delta(H),\delta(H)+1}$$

II.2.4.3 Cruzamiento uniforme

Sean A y B dos cadenas que se desean cruzar para formar una cadena C y una D. En este tipo de cruzamiento cada posición de bit de la cadena C es ordenada entre A y B. La que resulte elegida aportará su valor correspondiente a la misma posición para que sea colocado en C y la perdedora aportará su bit correspon-

diente a D. Supóngase que se desea preservar un esquema *H* del que es representante alguna de las cadenas padre. Este esquema posee *o*(*H*) bits fijos. No importa a cuál de los hijos vaya a dar el primer bit fijo de *H*. Lo que sí importa es que los restantes *o*(*H*)-1 bits del esquema vayan a dar exactamente al mismo. Cada uno de estos bits tiene probabilidad 1/2 de quedar en la misma cadena que el primero, así que la probabilidad de que los *o*(*H*) bits del esquema sean copiados a la misma cadena resultante es:

$$\left(\frac{1}{2}\right)^{o(H)-1}$$

Por lo tanto, la probabilidad de romper el esquema *H* es:

$$P_{\delta(H),o(H)} = 1-\left(\frac{1}{2}\right)^{o(H)-1}$$

Esto en el peor de los casos, cuando los padres no coinciden en ninguna de las posiciones de interés para el esquema. En general:

$$p_r \leq 1-\left(\frac{1}{2}\right)^{o(H)-1}$$

II.3 CRÍTICAS

Se han hecho diversas críticas al teorema del esquema (un panorama general de ellas se puede encontrar en [12]), las más importantes son:

a) Sólo es una cota inferior, es decir, no es exacto.
b) No es muy útil para predecir a largo plazo el comportamiento de un algoritmo genético.
c) Sólo considera los efectos destructivos de los operadores genéticos y no los efectos constructivos.
d) Es muy particular. Está hecho para un AGS con selección proporcional (de ruleta), cruzamiento de un punto y probabilidad de mutación uniforme.[3]

Sin embargo, el teorema del esquema ha sido durante años uno de los pocos intentos formales para modelar los algoritmos genéticos y ha servido de punto de partida para otros modelos más recientes.

[3] En la presentación hecha aquí se ha procurado ser más general, en la expresión que se presenta no aparece involucrado el tipo de cruzamiento.

II.4 PARALELISMO IMPLÍCITO Y LA HBC

El poder de los algoritmos genéticos reside en el hecho de que al procesar (evaluar) una sola cadena de una población, se evalúan, implícitamente, muchos esquemas en paralelo. Cada cadena binaria de longitud l es representante de 2^l esquemas (por cada una de las l posiciones se puede elegir el bit de la cadena o el *). Cada individuo de la población no es sólo el código de un elemento del dominio del problema sino que es, además, una muestra de varios subespacios de dicho dominio.

Supóngase que se tiene una población de N cadenas binarias de longitud l. Si todas las cadenas son iguales, entonces en esa población están representados sólo 2^l esquemas. Si todas las cadenas son diferentes podría ser que estuvieran representados hasta un máximo de $2^l N$ esquemas. Al procesar la población se estaría muestreando una cantidad mucho mayor de esquemas.

Las cantidades mencionadas son extremas. Corresponden al peor y al mejor de los casos, pero generalmente se tendrán esquemas con muchos representantes, otros con pocos o con ninguno, algunos se perderán al cruzar o mutar a los individuos. ¿Cuál es el número estimado de esquemas que son procesados?

Antes de responder a esta pregunta se procederá a contar el número de esquemas de un orden cualquiera que es posible definir en cadenas de longitud l. Claramente hay $\binom{l}{i}$ maneras de elegir i posiciones de un total de l, si en cada una de estas i posiciones es posible colocar un 1 ó un 0, entonces se tienen:

$$2^i \binom{l}{i}$$

posibles esquemas de orden i.

Para responder la pregunta formulada se seguirá el procedimiento mostrado en [12], que fue originalmente planteado por Fitzpatrick y Grefenstette [4]. Sea θ el orden más grande de un esquema representado en una población por, al menos, ψ individuos, y sea $N=2^{\theta}\psi$ el tamaño de la población. El número de diferentes esquemas de orden θ es:

$$2^{\theta}\binom{l}{\theta} = K$$

Si se establece: $\psi = 8$, $N=2^8$ y $l = 2^6$ se tiene que:

$$\theta = \log_2\left(\frac{2^8}{8}\right) = \log_2(2^5) = 5$$

de donde:

$$K = 2^5\binom{2^6}{5} = 32\left(\frac{64!}{5!59!}\right) = 243{,}984{,}384 > 16{,}777{,}216 = N^3$$

Por supuesto no es cierto que en general $K > N^3$. Dada una longitud de cadena l el número de esquemas posibles es finito (3^l) y el tamaño de la población puede elegirse arbitrariamente. Si se elige $N = 3^l$ entonces a lo más N esquemas serán muestreados. Pero N puede siempre ser elegido respecto a l de forma tal que resulte cierto que $K > N^3$. Fitzpatrick y Grefenstette suponen $L \geq 64$, $2^6 \leq N \leq 2^{20}$ y éste es un rango bastante amplio de valores prácticos. Existen resultados más generales como los exhibidos por Bertoni y Dorigo en [3].

En el teorema del esquema, los esquemas más pequeños (de longitud de definición corta respecto de la longitud total de la cadena) sobreviven más fácilmente que los esquemas grandes ante el operador de cruza. Además, la selección favorece esquemas con una alta calificación (de elevada aptitud). Este par de indicios lleva a pensar que las soluciones encontradas por un algoritmo genético se construyen a lo largo de generaciones mezclando esquemas cortos y de alto grado de adaptación. A éstos se les denomina *bloques constructivos* (*building blocks*), y a esta suposición la hipótesis de los bloques constructivos (HBC).

Hay bastante evidencia empírica que hace plausible la HBC. Sin embargo, también existen indicios que hacen pensar que no es verdadera, o al menos no en todos los casos (véase por ejemplo [10]). En general, el que la HBC sea cierta o no, depende del problema a resolver, así como de la codificación que se haya hecho del dominio. Un esquema de codificación en donde la interacción de los bits de la cadena genética (*epistasis*) es baja, favorece la propagación de bloques constructivos [1].

II.5 EXPLORACIÓN Y EXPLOTACIÓN

Supóngase que se está frente a una máquina de juego tragamonedas. Esta máquina es un poco peculiar, pues en vez de poseer sólo un brazo como las de Las Vegas, posee dos. Se tiene cierta cantidad de fichas para jugar (N). Cada vez que se introduce una ficha en la máquina el jugador puede elegir alguno de los brazos y accionarlo. Al hacer esto se reciben ciertos puntos. El objetivo es acumular la mayor cantidad posible de puntos en las N tiradas. Sean b_1 y b_2 los brazos. Cada uno de éstos proporciona una cantidad de puntos (premio) distribuida alrededor de cierta media y con cierta desviación estándar, sean (μ_1, σ_1) y (μ_2, σ_2) las medias y desviaciones respectivas de cada brazo. Estos valores no cambian con el tiempo y evidentemente el jugador los desconoce.

Dado que el jugador no posee información *a priori* acerca de cuál brazo es el mejor, deberá experimentar, resolver el problema sobre la marcha (*On-line*). Lo más razonable es muestrear ambos brazos y accionar más frecuentemente aquél

que proporcione un mayor valor medio de puntos observado. Es decir, se tiene que explorar ambos brazos y explotar el conocimiento que se vaya adquiriendo de esta exploración. Existe un compromiso intrínseco entre exploración y explotación: evidentemente la mejor manera de explorar ambos brazos es asignar a cada uno $N/2$ fichas. Pero eso no maximiza el premio acumulado. Se podría asignar una ficha a cada brazo y las restantes N-2 al que haya entregado el mayor premio en esa primera tirada. Pero de esta forma hay una alta probabilidad de que se le haya dado preferencia al brazo malo sólo porque en el primer intento el bueno no respondió como suele hacerlo.

¿De qué manera se deben asignar las fichas para maximizar el premio acumulado? Holland y otros antes de él estudiaron este problema. A continuación se presenta un esbozo de la solución planteada en [7]. Supóngase que se asignan en total n fichas al brazo con el menor premio observado (al que se denotará con b_m (N,n)), y por lo tanto N-n al de mayor premio observado (al que se denotará con b_b (N,N-n)). Se pretende encontrar el valor de n que minimiza las pérdidas ocasionadas por asignar fichas al brazo malo (este valor se denotará con n*). Sin menoscabo de generalidad supóngase que el brazo de premio más alto es en realidad b_1 y el de menor b_2 (i.e., $\mu_1 > \mu_2$). Hay dos maneras de desperdiciar fichas:

1. El brazo observado como el bueno (b_b(N,N-n)) es en realidad el malo (b_2), lo que ha hecho que el jugador desperdicie N-n fichas en él.
2. El brazo observado como bueno (b_b(N,N-n)) es en realidad el bueno (b_1), lo que ha ocasionado que se desperdicien n fichas en b_m(N,n).

El primer caso ocurre con cierta probabilidad, a la que se denotará con q y la pérdida de puntos esperada sería (N-n)(μ_1-μ_2). En el segundo caso, que por lo tanto ocurrirá con probabilidad (1-q), la pérdida esperada sería n(μ_1-μ_2). Así que la pérdida de puntos total esperada sería:

$$\begin{aligned} L(N - n, n) &= [q(N-n) + (1-q)n]\,(\mu_1-\mu_2) \\ &= (qN - qn + n - qn)\,(\mu_1-\mu_2) \\ &= (qN + n - 2qn)\,(\mu_1-\mu_2) \end{aligned} \qquad \text{(II.5.A)}$$

Derivando implícitamente II.5.A se obtiene:

$$\begin{aligned} \frac{dL}{dn} &= (\mu_1 - \mu_2)\left[N\frac{dq}{dn} + 1 - 2\left(q + n\frac{dq}{dn} \right) \right] \\ &= (\mu_1 - \mu_2)\left[N\frac{dq}{dn} + 1 - 2q - 2n\frac{dq}{dn} \right] \\ &= (\mu_1 - \mu_2)\left[(N - 2n)\frac{dq}{dn} + 1 - 2q \right] \end{aligned}$$

$$= (\mu_1 - \mu_2)\left(1 - 2q + (N - 2n)\frac{dq}{dn}\right)$$

Donde, como ya se dijo: $q = P(b_b(N,N\text{-}n) = b_2) = P(b_m(N,n) = b_1)$.
Para encontrar n* que minimiza II.5.A es necesario resolver:

$$\frac{dL}{dn} = (\mu_1 - \mu_2)\left(1 - 2q + (N - 2n)\frac{dq}{dn}\right) = 0 \qquad \text{(II.5.B)}$$

Para ello es menester expresar q en términos de n. Dado que $q = P(b_m(N, n) = b_1)$, supóngase que $b_m(N, n) = b_1$ y que por tanto, a b_1 se le asignaron n fichas y $N\text{-}n$ a b_2. Sean S_1^n y S_2^{N-n} los puntos acumulados en las n tiradas asignadas a b_1 y los acumulados en las $N\text{-}n$ asignadas a b_2, respectivamente. Entonces podemos escribir:

$$q = P\left(\frac{S_2^{N-n}}{N-n} > \frac{S_1^n}{n}\right) = P\left(\frac{S_1^n}{n} - \frac{S_2^{N-n}}{N-n} < 0\right) \qquad \text{(II.5.C)}$$

Pero por el teorema central del límite la distribución de $S_2^{N\text{-}n}/(N\text{-}n)$ puede ser aproximada por una distribución normal con media μ_2 y varianza $\sigma_2^2/(N\text{-}n)$ y S_1^n/n puede ser aproximada por una normal con media μ_1 y varianza σ_1^2/n. Así que por la convolución de distribuciones normales $S_1^n/n - S_2^{N\text{-}n}/(N\text{-}n)$ puede aproximarse por una distribución normal con media $\mu_1 - \mu_2$ y varianza $\sigma_1^2/n - \sigma_2^2/(N\text{-}n)$, es decir:

$$q(n) \approx \frac{1}{\sqrt{2\pi}}\frac{e^{\frac{-x^2}{2}}}{x}$$

donde:

$$x = \frac{\mu_1 - \mu_2}{\sqrt{\sigma_1^2 + \sigma_2^2}}\sqrt{n}$$

Utilizando esto ya es posible resolver II.5.B:

$$n^* \approx k^2 \ln\left(\frac{N^2}{8\pi k^4 \ln(N^2)}\right)$$

donde: $k = \sigma_1/(\mu_1 - \mu_2)$

Usando esta última expresión se obtiene:

$$N - n^* \approx \sqrt{8\pi k^4 \ln(N^2)}\; e^{\frac{n^*}{2k^2}}$$

En síntesis, el mecanismo para lograr el mayor número de puntos posible es asignar exponencialmente más fichas al mejor brazo observado que las que se asignan al peor.

Probablemente, en este momento el lector se estará preguntando: ¿Qué tiene que ver esto con los algoritmos genéticos? Supóngase que un esquema en particular (*H*) tiene una calificación media $\overline{f} + c\,\overline{f}$, donde $c > 1$ y $\overline{f}$ es el promedio de calificación de la población, es decir, *H* tiene una calificación por arriba del promedio de la población. Si se reescribe el teorema del esquema (considerando únicamente selección proporcional), con esta suposición se obtiene:

$$m(H, t + sel) = m(H, t+1) = m(H, t)\frac{\overline{f} + c\overline{f}}{\overline{f}} = (1+c)\,m(H, t)$$

Si se supone que el valor de *c* es estacionario (no varía de generación en generación) se obtiene:

$$m(H, t) = m(H, 0)\,(1+c)^t \qquad \text{(II.5.D)}$$

donde $m(H,0)$ es el número de representantes del esquema *H* al tiempo $t=0$. Lo que dice la ecuación II.5.D es que el proceso genético (considerando sólo la selección) incrementa exponencialmente con el tiempo el número de representantes de los esquemas buenos (que están por encima del promedio).

Esto quiere decir que el proceso genético implícitamente resuelve el problema de una máquina tragamonedas con 2^r brazos [8]. Se muestrean los 2^r esquemas de orden *r* y se incrementa exponencialmente el número de muestras asignadas al mejor de éstos respecto del segundo mejor, y así sucesivamente hasta llegar al menos apto. Esto hace del algoritmo genético un mecanismo de optimización muy útil para aplicaciones que requieren de un buen desempeño *On-line*, tales como aplicaciones de control en tiempo real y, en general, para resolver problemas de índole dinámica, donde la bondad de cada posible solución probada cuenta en el desempeño global. En contraste, están los problemas estáticos, donde lo que cuenta es el desempeño *Off-line*, es decir, si la solución correcta es encontrada o no y en cuánto tiempo, sin importar los intentos previos. En síntesis, un algoritmo genético no es, en sí, un optimizador de funciones, más bien, como señala Mitchell [8], es un "maximizador de ganancia acumulada". El objetivo no es encontrar un punto óptimo, sino maximizar la ganancia que se logra al

tiempo que se explora el espacio de búsqueda, muestreando diversas posibilidades.

II.6 ENGAÑOS Y RESULTADOS INESPERADOS

Como se mencionó, el alcance del teorema del esquema para explicar el comportamiento de un algoritmo genético es bastante limitado. No puede utilizarse para decidir bajo qué circunstancias es fácil o difícil resolver un problema mediante el uso de algoritmos genéticos. En esta sección se presentarán dos análisis elaborados con esta finalidad. En el primero de ellos se pretende determinar cuáles son los problemas difíciles más pequeños que se pueden definir y, en el segundo, se investiga con más detalle el procesamiento de esquemas tras definir un tipo de problemas que favorecen la formación de bloques constructivos.

II.6.1 Problemas engañosos mínimos

II.6.1.1 Los tipos de problemas engañosos mínimos

Supóngase que se tiene un AG que manipula cadenas de longitud 8 y que se tienen cuatro esquemas de orden dos con sus respectivos valores de adaptación promedio, tal como se muestra en la figura F.II.6.A.

Supóngase que los valores de adaptación son constantes y que:

$$f_{11} > f_{00} \qquad \text{(II.6.A)}$$
$$f_{11} > f_{01} \qquad \text{(II.6.B)}$$
$$f_{11} > f_{10} \qquad \text{(II.6.C)}$$

el máximo global se encuentra en el subespacio definido por 1*****1*.

Sean:

$$\overline{f}(0*) = \frac{f_{00} + f_{01}}{2} \qquad \overline{f}(*0) = \frac{f_{00} + f_{10}}{2}$$
$$\overline{f}(1*) = \frac{f_{11} + f_{10}}{2} \qquad \overline{f}(*1) = \frac{f_{11} + f_{01}}{2}$$

Para tratar de engañar al AG tendría que ocurrir:

$$\overline{f}(0*) > \overline{f}(1*) \qquad \text{(II.6.D)}$$

o bien:

$$\overline{f}(*0) > \overline{f}(*1) \qquad \text{(II.6.E)}$$

esto es, que en promedio los esquemas de orden 1 que no contienen el máximo global sean mejores que aquellos que lo contienen. Las pistas que puede seguir el AG para encontrar el máximo global lo llevan lejos de él, le indican que muestree con mayor frecuencia subespacios que no lo contienen.

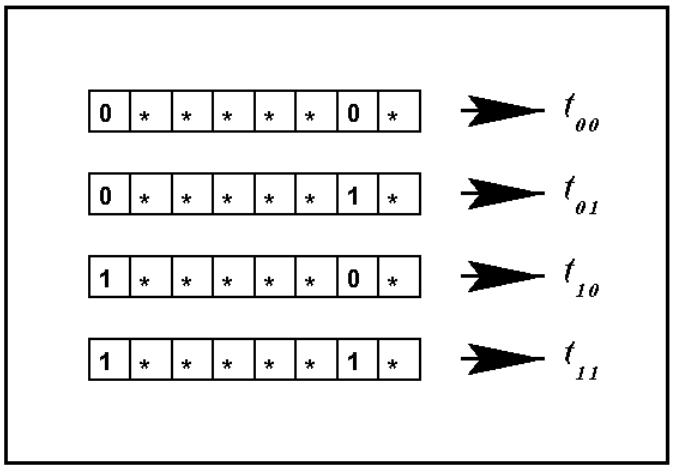


FIGURA F.II.6.A: *Cuatro esquemas de orden 2 y su adaptación promedio.*

Las expresiones II.6.D y II.6.E no pueden ocurrir simultáneamente. Esto puede demostrarse fácilmente por reducción al absurdo. Supóngase que son verdaderas ambas, es decir, ocurre:

$$\frac{f_{00} + f_{01}}{2} > \frac{f_{11} + f_{10}}{2}$$

y además:

$$\frac{f_{00} + f_{10}}{2} > \frac{f_{11} + f_{01}}{2}$$

sumando los lados izquierdos y derechos de ambas expresiones y multiplicando por dos se obtiene:

$$2 f_{00} + f_{01} + f_{10} > 2 f_{11} + f_{10} + f_{01}$$

de donde:

$$f_{00} > f_{11}$$

lo que contradice la suposición inicial II.6.A.

Supóngase entonces, sin pérdida de generalidad, que ocurre II.6.D, es decir:

$$f_{11} + f_{10} < f_{00} + f_{01}$$

de donde:

$$f_{11} < f_{00} + f_{01} - f_{10} \qquad \text{(II.6.F)}$$

Se tienen hasta ahora definidas tres relaciones II.6.A, II.6.B y II.6.C entre los cuatro valores medios de f (f_{11}, f_{10}, f_{01} y f_{00}) esto significa que faltan aún por definir otras tres relaciones entre ellos (en total hay $\binom{4}{2} = 6$) para tener completamente especificado el problema.

Considerando II.6.A y II.6.F se obtiene:

$$f_{00} < f_{11} < f_{00} + f_{01} - f_{10}$$
$$f_{00} < f_{00} + f_{01} - f_{10}$$
$$0 < f_{01} - f_{10}$$

así que:

$$f_{10} < f_{01} \qquad \text{(II.6.G)}$$

Considerando II.6.B y II.6.F resulta:

$$f_{01} < f_{11} < f_{00} + f_{01} - f_{10}$$
$$f_{01} < f_{00} + f_{01} - f_{10}$$
$$0 < f_{00} - f_{10}$$

de donde:

$$f_{10} < f_{00} \qquad \text{(II.6.H)}$$

Sólo falta por definir la relación entre f_{01} y f_{00}, y es posible elegirla de dos maneras diferentes ya que no hay ninguna otra restricción que determine la relación entre ambas. Entonces se tienen dos tipos posibles de problemas engañosos diferentes:

Tipo I: $f_{01} > f_{00}$ (II.6.I)

Tipo II: $f_{01} \leq f_{00}$ (II.6.J)

Los dos tipos de problemas engañosos están ilustrados en las figuras F.II.6.B 1 y 2.

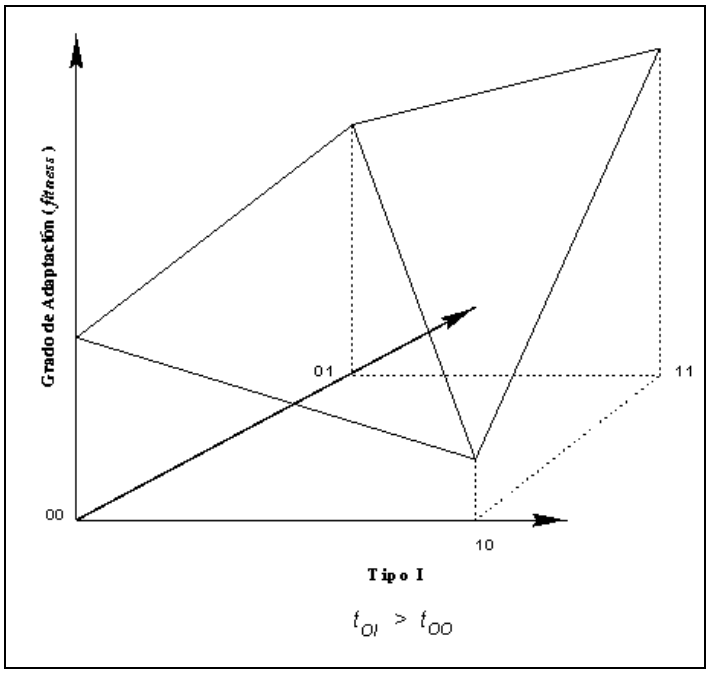


FIGURA F.II.6.B.1: *Problema engañoso mínimo de Tipo I.*

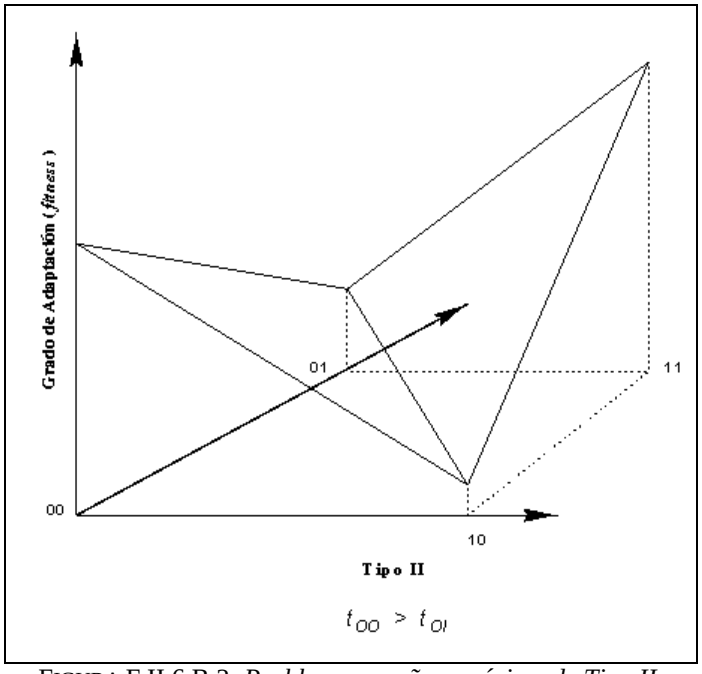


FIGURA F.II.6.B.2: *Problema engañoso mínimo de Tipo II.*

II.6.1.2 Análisis de los problemas engañosos

Se procederá ahora a hacer un análisis del comportamiento de un AG sin mutaciones, tratando de resolver problemas engañosos. Se utilizará para ello el teorema del esquema. Recuérdese la expresión encontrada para éste pero excluyendo las mutaciones:

$$m(H,t+1) \geq m(H,t)\frac{\overline{f}(H)}{\overline{f}}(1-p_c p_r)$$

Si se utiliza cruzamiento de un solo punto, reemplazando p_r por la expresión correspondiente a la probabilidad de ruptura del esquema *H*, se obtiene:

$$m(H,t+1) \geq m(H,t)\frac{\overline{f}(H)}{\overline{f}}\left(1-p_c\frac{\delta(H)}{l-1}\right)$$

donde $\delta(H)$ denota la longitud de definición de *H* y *l* la longitud total de las cadenas manipuladas por el algoritmo.

Sea $P_c = \left(p_c\frac{\delta(H)}{l-1}\right)$, la probabilidad de cruzar y romper el esquema *H*, en el caso presentado en la sección anterior todos los esquemas tenían la misma longitud de definición, así que P_c es la misma para todos ellos. La cantidad de representantes del esquema en la generación *t*+1 se incrementa en un factor:

$$\beta_H = \frac{\overline{f}(H)}{\overline{f}}(1-P_c)$$

Evidentemente, si $\beta_H > 1$ el número de representantes de *H* se incrementa, si $\beta_H < 1$ decrece y si $\beta_H = 1$ se mantiene igual. Así que se esperaría que en un problema engañoso ocurriera que $\beta_H \leq 1$. En el caso presentado en la sección anterior se esperaría que:

$$\beta_{11} = \frac{\overline{f}(11)}{\overline{f}}\left(1-p_c\frac{\delta(11)}{l-1}\right) = \frac{\overline{f}(11)}{\overline{f}}(1-P_c) \leq 1$$

Se tendrá que hacer ahora un análisis más detallado que cuando se presentó el teorema del esquema. Recuérdese que éste es sólo una cota inferior y no proporciona la cantidad exacta de representantes de un esquema determinado. En el caso que se presenta ahora se deberán considerar también aquellos representantes del esquema *H* generados a partir del cruce de dos individuos que no eran representantes de dicho esquema. Para poder hacer esto se debe elaborar una

tabla donde se indique cuáles son las posibles cadenas que se generan a partir de una pareja determinada. Esto aparece en la tabla T.II.6.A, el análisis es similar al que aparece en [6]. Un "-" significa que al cruzar un representante del esquema que aparece en la fila con uno del que aparece en la columna el resultado es, nuevamente, representante de alguno de ellos.

TABLA T.II.6.A: *Análisis de cruces de los esquemas.*

Cruce	00	01	10	11
00	-	-	-	01, 10
01	-	-	00, 11	-
10	-	00, 11	-	-
11	10, 01	-	-	-

Como en [6], P_H^t expresa la probabilidad de encontrar en la generación *t* el esquema *H*, es decir, la proporción de la población al tiempo *t* que es representante de *H*; con $\overline{f}^t$ se representa el valor esperado de adaptación (*fitness*) en la generación al tiempo *t*, y con f_H se representa la adaptación promedio de los representantes del esquema *H*. De hecho:

$$\overline{f}^t = P_{00}^t f_{00} + P_{01}^t f_{01} + P_{10}^t f_{10} + P_{11}^t f_{11}$$

por definición de valor esperado ($E(X) = \sum_{X=x} xp(x)$).

Considerando esto, es posible calcular la probabilidad de encontrar el esquema 11 en la generación al tiempo *t*+1 si se conoce la probabilidad de hallarlo en la del tiempo *t*. De la tabla se observa que el esquema 11 se rompe cuando un representante de éste es cruzado con uno del esquema 00, así que el esquema 11 se rompe cuando es seleccionado para cruzarse y el punto de corte cae entre los bits definidos, y el seleccionado para cruzarse con éste es el 00. Esto queda completamente expresado en términos de probabilidad como:

$$p_c \frac{\delta(11)}{l-1} \frac{f_{00}}{\overline{f}^t} P_{00}^t = P_c \frac{f_{00}}{\overline{f}^t} P_{00}^t$$

Por otra parte, el esquema 11 es generado cuando se cruzan representantes del esquema 10 y del 01 (véase la tabla) y su punto de corte se selecciona entre los bits definidos, es decir, la probabilidad de que el 11 sea generado con cruzamiento es:

$$P_c \frac{f_{01}}{\overline{f}^t} P_{01}^t \frac{f_{01}}{\overline{f}^t} P_{10}^t = P_c \frac{f_{01} f_{10}}{\left(\overline{f}^t\right)^2} P_{01}^t P_{10}^t$$

En resumen, la probabilidad de aparición del esquema 11 en la generación al tiempo t+1 es:

$$P_{11}^{t+1} = P_{11}^t \frac{f_{11}}{\overline{f}^t}\left(1 - P_c \frac{f_{00}}{\overline{f}^t} P_{00}^t\right) + P_c \frac{f_{01} f_{10}}{\left(\overline{f}^t\right)^2} P_{01}^t P_{10}^t \qquad \text{(II.6.K)}$$

De manera análoga, para los demás esquemas:

$$P_{10}^{t+1} = P_{10}^t \frac{f_{10}}{\overline{f}^t}\left(1 - P_c \frac{f_{01}}{\overline{f}^t} P_{01}^t\right) + P_c \frac{f_{00} f_{11}}{\left(\overline{f}^t\right)^2} P_{00}^t P_{11}^t \qquad \text{(II.6.L)}$$

$$P_{01}^{t+1} = P_{01}^t \frac{f_{01}}{\overline{f}^t}\left(1 - P_c \frac{f_{10}}{\overline{f}^t} P_{10}^t\right) + P_c \frac{f_{00} f_{11}}{\left(\overline{f}^t\right)^2} P_{00}^t P_{11}^t \qquad \text{(II.6.M)}$$

$$P_{00}^{t+1} = P_{00}^t \frac{f_{00}}{\overline{f}^t}\left(1 - P_c \frac{f_{11}}{\overline{f}^t} P_{11}^t\right) + P_c \frac{f_{01} f_{10}}{\left(\overline{f}^t\right)^2} P_{01}^t P_{10}^t \qquad \text{(II.6.N)}$$

Que el algoritmo genético converja significa que:

$$\lim_{t\to\infty} P_{11}^t = 1$$

se esperaría que en los problemas engañosos esto no ocurriera u ocurriera con dificultad.

Mediante las ecuaciones II.6.K–II.6.N es posible predecir la proporción de cada esquema en una población, dadas ciertas condiciones iniciales. En las figuras F.II.6.C–F.II.6.F se muestran los resultados de iterar este sistema de recurrencias. Los valores de adaptación promedio de cada esquema aparecen al pie de cada figura. En todos los casos la probabilidad de cruza es de 0.6, la longitud de cadena es de 8 y la longitud de definición de los esquemas es de 6. Nótese que es determinante la proporción inicial de esquemas en la convergencia del algoritmo. En las gráficas F.II.6.D y F.II.6.F los cuatro esquemas son, al inicio, equiprobables (0.25) y en estos casos el algoritmo es engañado sólo al principio (figura F.II.6.D) o nunca (figura F.II.6.F), mientras que en los casos donde las proporciones iniciales de los esquemas no son iguales (P_{00}^0 =0.7,

$P_{01}^0 = P_{10}^0 = P_{11}^0 = 0.1$) el algoritmo es engañado durante muchas generaciones (figura F.II.6.C) o definitivamente no converge al óptimo (figura F.II.6.E). Cabe hacer notar que en la figura F.II.6.E mueren muy rápido los esquemas 11 (el óptimo) y el 10. Sin este último es imposible generar nuevamente el 11 mediante cruzamientos, por lo que se extingue irremediablemente.

Los resultados mostrados fueron calculados mediante un pequeño programa en C. El código de las dos funciones principales de éste aparece en la figura F.II.6.G. `valesp` calcula el valor esperado de la adaptación dadas las proporciones de cada esquema y sus grados de adaptación, `calcula` efectúa una iteración del sistema II.6.K–II.6.N.

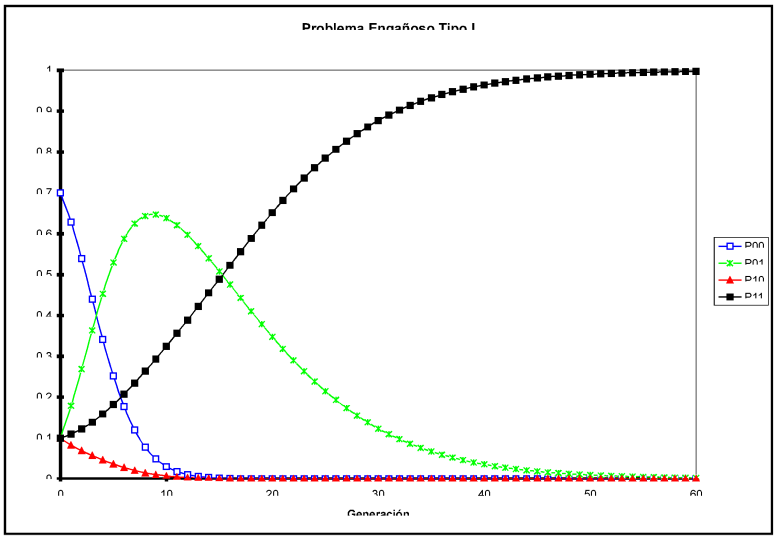


FIGURA F.II.6.C: *Problema engañoso de tipo I:* $f_{00}=5$, $f_{01}=7$, $f_{10}=2$ *y* $f_{11}=8$.

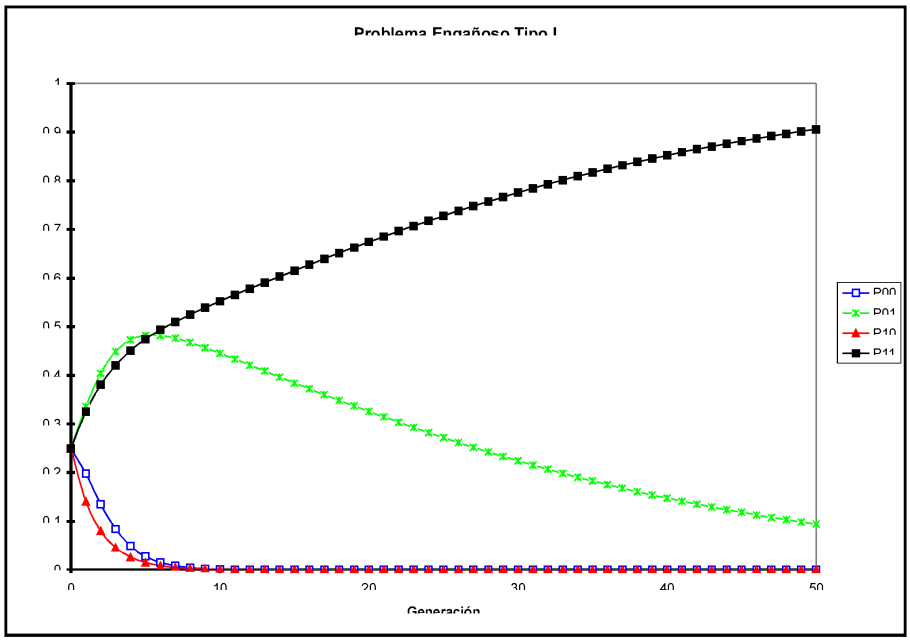


FIGURA F.II.6.D: *Problema engañoso de tipo I:* $f_{00}=5$, $f_{01}=7.6$, $f_{10}=3$ *y* $f_{11}=8$.

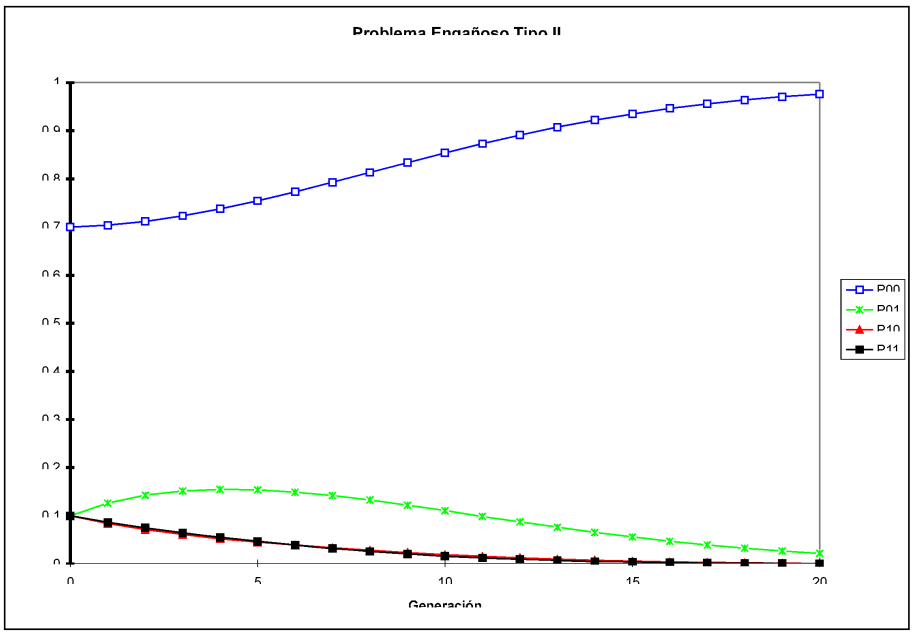


FIGURA F.II.6.E: *Problema engañoso de tipo II:* $f_{00}=5$, $f_{01}=4$, $f_{10}=2$ *y* $f_{11}=6$.

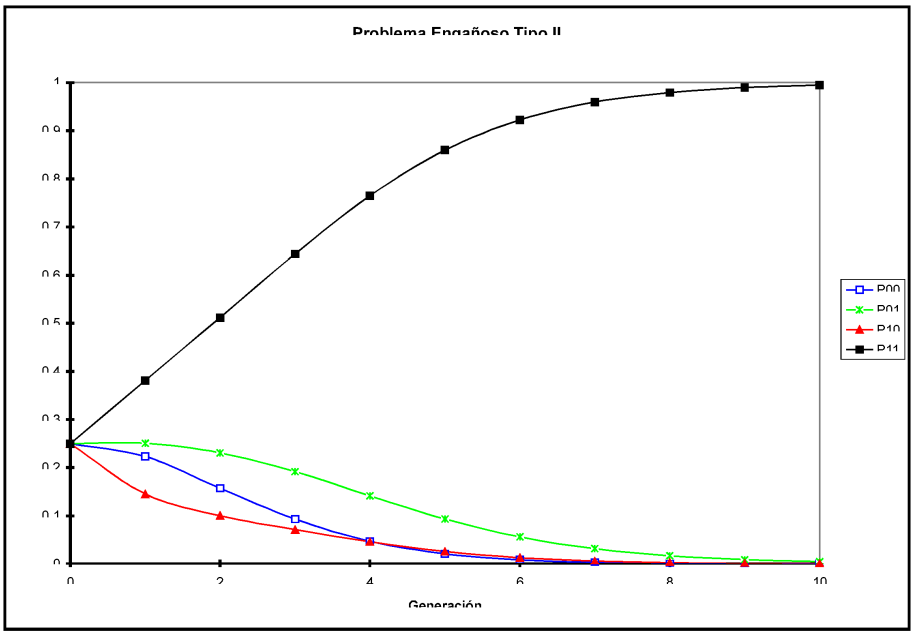


FIGURA F.II.6.F: *Problema engañoso de tipo II:* $f_{00}=5$, $f_{01}=4$, $f_{10}=2$ *y* $f_{11}=8$.

II.6.2 Caminos regios

II.6.2.1 Definición del problema

El análisis y los resultados presentados en esta sección fueron reportados por Melanie Mitchell, John Holland y Stephanie Forrest en [9] y [5], y aparecen también en [8]. La intención era la de investigar el procesamiento de esquemas y la recombinación en algoritmos genéticos simples. Para ello se definieron algunos problemas cuya función de adaptación[4] posee algunas características que favorecen la hipótesis de los bloques constructivos. Las características principales de los caminos regios son:

1. Existencia de bloques pequeños de alto grado de adaptación.
2. Existencia de escalones. Esquemas de orden intermedio con grado de adaptación mayor que los mencionados en el inciso anterior, pero que resultan de combinar éstos y que son capaces de generar a, su vez, esquemas aún mejores.

[4] De hecho, la forma de dicha función, lo que en inglés suele llamarse *fitness landscape*.

```
void calcula (float fi00, float fi01, float fi10, float fi11, float prc,
              int del, int lon, float *Pr00, float *Pr01, float *Pr10,
              float *Pr11)
     {
        float pp1, pp2, pp3, ve; float tp00, tp01, tp10, tp11;

        pp1 = prc*((float)del/(float)lon);
        ve  = valesp (fi00, fi01, fi10, fi11,*Pr00,*Pr01,*Pr10,*Pr11);

        pp2  = 1.0 - (pp1 *(fi00/ve)*(*Pr00));
        pp3  = pp1*((fi01*fi10)/(ve*ve))*(*Pr01)*(*Pr10);
        tp11 = (*Pr11) * (fi11/ve) * pp2 + pp3;

        pp2  = 1.0 - (pp1 *(fi11/ve)*(*Pr11));
        tp00 = (*Pr00) * (fi00/ve) * pp2 + pp3;

        pp2  = 1.0 - (pp1 *(fi01/ve)*(*Pr01));
        pp3  = pp1*((fi00*fi11)/(ve*ve))*(*Pr00)*(*Pr11);
        tp10 = (*Pr10) * (fi10/ve) * pp2 + pp3;

        pp2  = 1.0 - (pp1 *(fi10/ve)*(*Pr10));
        tp01 = (*Pr01) * (fi01/ve) * pp2 + pp3;

        *Pr00 = tp00; *Pr01 = tp01;
        *Pr10 = tp10; *Pr11 = tp11;
     }

     float valesp (float fi00, float fi01, float fi10, float fi11,
                   float Pr00, float Pr01, float Pr10, float Pr11)
     {
        float sum;
        sum = 0;
        sum = fi00*Pr00 + fi01*Pr01 + fi10*Pr10 + fi11*Pr11;
        return (sum);
     }
```

FIGURA F.II.6.G: *Fragmento del código en C que implanta las ecuaciones II.6.K–II.6.N.*

Una de las funciones definidas por Mitchell *et al.* es la siguiente:

$$R_1(x) = \sum_{i=1}^{8} c_i \delta_i(x)$$

donde:

$$\delta_i(x) = \begin{cases} 1 & si\ x \in S_i \\ 0 & en\ otro\ caso \end{cases}$$

En estas expresiones S_i son los esquemas mostrados en la figura F.II.6.H, $x \in S_i$ significa que la cadena binaria x es representante del esquema S_i y c_i son los coeficientes asignados a cada uno de ellos (de hecho a cada S_i se le asigna su orden), en este caso c_i=8 para toda $i \in 1,...,8$, así que se puede reescribir R_1 como:

$$R_1(x) = 8\sum_{i=1}^{8} \delta_i(x)$$

La función $\delta_i(x)$ dice si una cadena dada *x* es representante o no del esquema S_i. Así que, básicamente, la función $R_1(x)$ es multiplicar por 8 el número de esquemas de los que es representante la cadena *x*. Por ejemplo, si

$$x = 11111111000000000111000...00011111111$$

entonces $R_1(x) = 8 \bullet \quad 2 = 16$. Evidentemente la cadena binaria en la que $R_1(x)$ obtiene su valor máximo es la que está constituida por 64 unos. Ésta también aparece en la figura F.II.6.H denotada por S_{max}.

En los experimentos realizados por Mitchell se utilizó un esquema de selección llamado de truncamiento sigma (*sigma truncation*). Bajo éste, el número esperado de copias de un individuo con adaptación F_i es:

$$\frac{F_i - \overline{F}}{2\sigma} + 1$$

donde $\overline{F}$ denota el valor promedio de adaptación en la población y σ su desviación estándar. Adicionalmente, el número de copias no podía exceder de 1.5. Si el valor calculado con la expresión anterior sobrepasaba este número se establecía en 1.5. La intención al hacer esto fue retrasar la convergencia, de tal forma que no dominara rápidamente alguna cadena de alto grado de adaptación antes de poder descubrir alguna mejor. La probabilidad de cruza se estableció en 0.7 y la de mutación en 0.005. Se utilizó cruzamiento de un punto.

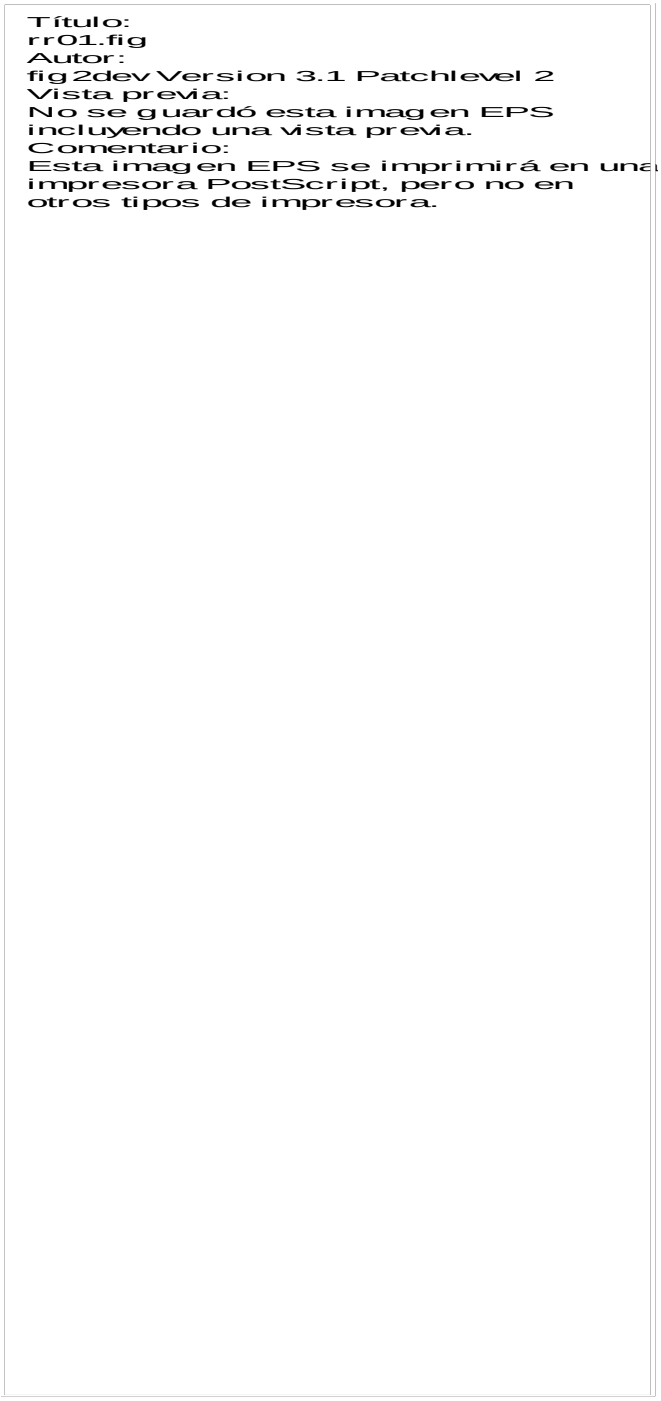


FIGURA F.II.6.H: *Los bloques constructivos de una cadena que maximiza la función camino regio utilizada como ejemplo.*

II.6.2.1 Escaladores

Con el fin de comparar el desempeño del algoritmo genético simple que opera sobre R_1, se eligieron otros algoritmos para resolver el mismo problema. Éstos fueron tres diferentes algoritmos de "escalada": escalada de ascenso de máxima pendiente (*steepest-ascent hill climbing*), escalada llegando al apoyo más próximo (*next-ascent hill climbing*) y escalada por mutación aleatoria (*random-mutation hill climbing*). A cada uno de estos algoritmos se les denotará por sus siglas en inglés: SAHC, NAHC y RMHC, respectivamente.

Los algoritmos se presentan a continuación. En éstos, *selecc* denota una función que regresa una cadena seleccionada aleatoriamente del dominio de búsqueda, $s[i]$ denota el i-ésimo alelo (bit) de izquierda a derecha de la cadena s, $f(x)$ denota el valor de la función de adaptación (*fitness*) evaluado en la cadena x, $\sim q$ denota la negación del bit q y la función posición regresa un número entero aleatorio uniforme en $\{1,2,...,l\}$, se supone que todas las cadenas tienen una longitud l y que se hacen, a lo más, E evaluaciones de la función de adaptación.

ESCALADA DE ASCENSO DE MÁXIMA PENDIENTE (SAHC):

1. Elegir aleatoriamente una cadena del dominio de búsqueda. Llamarla la *cima actual*.
2. Recorrer la cadena de izquierda a derecha invirtiendo cada bit, evaluando y recordando el grado de adaptación de la cadena resultante.
3. Si alguna de las cadenas generadas posee un grado de adaptación mayor que los obtenidos hasta ese momento, etiquetarla como la *cima actual*.
4. Si no hay incrementos en el grado de adaptación, guardar la *cima actual* e ir al paso 1, si no, ir al paso 2 con la nueva *cima actual*.
5. Cuando se hayan efectuado un total de E evaluaciones, regresar la *cima actual* encontrada hasta ahora.

```
PROCESS (SAHC)
begin
  fmx = -∞
  evals = 0
  repeat
    htop = selecc
    maxl = htop
    curr = htop
    for i=1 to l do
      curr[i] = ~htop[i]
      if (f(curr) > f(maxl))
      then
        maxl = curr
      endif
      evals = evals + 1
```

```
    endfor
    if (f(maxl) > fmx)
    then
      maxm = maxl
      fmx  = f(maxm)
    endif
  until (evals == E)
  return (maxm)
end
```

ESCALADA LLEGANDO AL APOYO MÁS PRÓXIMO (NAHC)

1. Elegir aleatoriamente una cadena del dominio de búsqueda. Llamarla la *cima actual*.
2. Para cada uno de los bits de la cadena: invertir el bit, si la cadena resultante tiene un grado de adaptación mayor que la *cima actual*, etiquetarla como la nueva *cima actual*, si no, regresar el bit a su valor original.
3. Si no se encontró incremento en el grado de adaptación, guardar la *cima actual* e ir al paso 1.
4. Cuando se hayan efectuado un total de *E* evaluaciones, regresar la *cima actual* encontrada hasta ahora.

```
PROCESS (NAHC)
begin
 fmx   = -∞
 evals = 0
 repeat
  htop = selecc
  ante = htop
  curr = htop
  for i=1 to l do
   curr[i] = ~ante[i]
   if (f(curr) > f(ante))
   then
     ante = curr
   else
     curr = ante
   endif
     evals = evals + 1
  endfor
  if (f(curr) > fmx)
  then
    maxm = curr
    fmx  = f(maxm)
  endif
```

```
  until (evals == E)
  return (maxm)
end
```

ESCALADA POR MUTACIÓN ALEATORIA (RMHC)

1. Elegir aleatoriamente una cadena del dominio de búsqueda. Llamarla la *cima actual*.
2. Elegir aleatoriamente una posición de la cadena (locus) e invertir el bit en esa posición. Si la cadena resultante posee un grado de adaptación mayor o igual al de la *cima actual*, etiquetarla como la nueva *cima actual*.
3. Ir al paso 2 hasta que haya sido encontrada la cadena de mejor adaptación o hasta que se hayan hecho *E* evaluaciones de la función de adaptación.
4. Cuando se hayan efectuado un total de *E* evaluaciones, regresar la *cima actual* encontrada hasta ahora.

```
PROCESS (RMHC)
begin
  evals = 0
  curr  = selecc
  maxm  = curr
  repeat
    locus = posicion
    curr[locus] = ~curr[locus]
    if (f(curr) ≥ f(maxm))
    then
      maxm = curr
    endif
    evals = evals + 1
  until (evals == E)
  return (maxm)
end
```

Resultados

Los resultados obtenidos por Mitchell son contrarios a lo que se esperaría si la HBC es correcta. El lector puede observarlos en la tabla T.II.6.B. Los números que aparecen entre paréntesis son los errores estándar (la desviación estándar dividida por el número de iteraciones). Se puede observar que el algoritmo que resultó mejor es el RMHC (es el que alcanza el óptimo en menos tiempo). Resulta además que al compararlo con el algoritmo genético (columna etiquetada con AG en la tabla), RMHC posee un error estándar menor y encuentra la solución en un número de iteraciones que es ¡un orden de magnitud inferior al del

AG! Esto resulta extraño si se piensa en que el problema fue hecho para favorecer el funcionamiento óptimo del AG.

TABLA T.II.6.B: *Resultados de los experimentos de Mitchell.*

200 iteraciones	AG	SAHC	NAHC	RMHC
Media	61,334 (2304)	>256,000 (0)	> 256,000 (0)	6,179 (186)
Mediana	54,208	>256,000	> 256,000	5,775

Una de las razones para que ocurra lo antes mencionado es lo que se ha dado en llamar *hitchhiking*, o correlación espuria. Esto se refiere a que, una vez que ha sido encontrado un representante con alta calificación de un esquema de orden alto, este esquema se propaga copiosamente y con rapidez en la población, con 0's en las demás posiciones. Esto ocasiona que se disminuya la proporción de otros esquemas que son necesarios para la construcción de la solución correcta.

Supóngase que al inicio de la ejecución del algoritmo se descubren algunos representantes de S_1 y de S_2 y que éstos engendran representantes de ambos esquemas simultáneamente, es decir, tienen un grado de adaptación de 16. Estos individuos se reproducen mucho más que el resto de la población dado que son seleccionados frecuentemente, por su alto grado de adaptación, y con muy alta probabilidad transmiten sus alelos a su descendencia. El algoritmo es ciego, así que junto con los valores en 1 (que son buenos en este caso) se van algunos en 0 cercanos a ésos (es más probable que se transmitan los cercanos que los lejanos, recuérdese el teorema del esquema). Como resultado de esto, los representantes de S_1 y S_2 proliferan en la población (hasta niveles del 90% o más, como prueban los experimentos de Mitchell) y se reduce la proporción de representantes de otros esquemas. Esto ocasiona que sea muy difícil encontrar esquemas mejores (no hay mucha variedad para cruzarse) especialmente aquellos que tengan 1's en posiciones cercanas a los 1's de S_1 y S_2 (porque se acarrean los 0's de los representantes de ellos que se han cruzado).

II.7 EJERCICIOS

1. Considere los siguientes esquemas:

   a) 101*1**1
   b) 0111***1

Calcule la probabilidad de que sobrevivan al cruzarse entre sí, si se utiliza el esquema de cruzamiento:
a) de un punto. *b)* de dos puntos. *c)* uniforme

2. Sean A y B dos cadenas binarias de longitud *l* y sea *H (A, B)* el número de bits en los que A y B difieren (distancia de Hamming). Demuestre que el número de posibles cadenas que se obtienen cruzando A y B (sin considerar a las mismas A y B) es:

   $2^{H(A,B)} - 2$, en el caso de cruzamiento uniforme
   $2\,(H(A,B) - 1)$, en el caso de cruzamiento de un punto
   $2\binom{H(A,B)}{2}$, en el caso de cruzamiento de dos puntos

   Tome en consideración que cada cruzamiento produce dos cadenas. (Sugerencia: consulte [12], pág. 18)

3. Sean A y B dos cadenas binarias de longitud *l* y sea *H (A, B)* la distancia de Hamming entre ellas (véase ejercicio anterior). ¿Cuál es la probabilidad de que A sea transformada en B mediante mutaciones? ¿Cuál es la probabilidad de que A se transforme en alguna cadena que diste de A en *H (A, B)* bits? Verifique su respuesta, la suma de las probabilidades de todos los eventos posibles debe ser 1. (Sugerencia: utilice el teorema del binomio.)

4. La versión más general y completa del teorema del esquema, sin considerar el operador de mutación, se puede expresar así (en términos de probabilidades):

$$P(Z,t+1) = P(Z,t)\frac{f(Z,t)}{\bar{f}} \quad \left[1-(p_c\ \textit{pérdida})\right] \quad +(p_c\ \textit{ganancia})$$

   el factor *pérdida* es la probabilidad de que al cruzar un representante de *Z* con alguna otra cadena, ninguno de los dos hijos sea representante de *Z*. El factor *ganancia* es la probabilidad de que dados dos individuos que no son representantes de *Z*, algunos de los hijos resultado de su cruzamiento sí lo sea.

   Considere que se manipulan cadenas binarias de longitud 3 y sea $Z = 100$. Supóngase que se conocen: *P(000, t), P(001, t), ..., P(111, t)*, así como: *f (000, t), f (001, t), ..., f (111, t)*. Calcule *P(100, t+1)*. (Sugerencia: consulte [12], pág. 23)

5. Supóngase que se utiliza un alfabeto de cardinalidad $k = 3$ cuyos símbolos son {0, 1, 2} y que se utilizan cadenas de longitud *l*. ¿Cuántas cadenas son

representantes de un esquema dado? ¿Cuántos esquemas diferentes existen? Resuelva ahora para cualquier número natural *k*.

6. Supóngase que se tiene un conjunto de *N* individuos representantes de un esquema H y que se está utilizando un alfabeto de cardinalidad *k*. Tomando en consideración lo analizado en la pregunta anterior, ¿puede elaborar una justificación técnica para el uso de preferente de un alfabeto binario?

7. Derive la expresión para el número total de posibles poblaciones de tamaño *N* de cadenas binarias de longitud *l*:

$$\Pi = \binom{N + 2^l - 1}{2^l - 1} = \binom{N + 2^l - 1}{N}$$

8. Supóngase que se tiene un problema donde se consideran los siguientes esquemas de orden 3 con sus respectivas calificaciones promedio:

Esquema	Calif. promedio
0**00	f_0
0**01	f_1
0**10	f_2
0**11	f_3
1**00	f_4
1**01	f_5
1**10	f_6
1**11	f_7

El máximo global es la cadena 01110. ¿Qué condiciones se deben cumplir entre f_0, f_1, ...,f_7 para que el problema sea engañoso? Examine todas las posibilidades y descubra cuáles se excluyen mutuamente, si es el caso.

II.8 PROGRAMACIÓN

1. Utilizando el código mostrado en la figura F.II.6.G, busque otros problemas engañosos mínimos variando los valores de las calificaciones promedio y las proporciones iniciales.

2. Programe el escalador de ascenso por mutación aleatoria y compare su desempeño con el del AGS en otros problemas. Para medir el desempeño de un AG puede utilizar el desempeño *On-line* (calificación promedio de todos los individuos explorados hasta la generación actual) y el desempeño *Off-line* (calificación promedio de los mejores individuos de cada generación). En el caso del RMHC ambas medidas de desempeño se convierten en la misma.

Incorpore un RMHC en su AGS y programe un mecanismo que haga que el escalador se invoque cada vez que el AGS se encuentre "estancado". Invente problemas difíciles, en los que este estancamiento ocurra y compare el desempeño del AGS con el del algoritmo genético auxiliado por el escalador. Reporte sus resultados.

II.9 REFERENCIAS

[1] Beasley, D., D. Bull y R. Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals", *University Computing*, 15(2), 1993, pp. 58-69.

[2] Beasley, D., D. Bull y R. Martin, "An Overview of Genetic Algorithms: Part 2, Research Topics", *University Computing*, 15(4), 1993, pp. 170-181.

[3] Bertoni, A., y M. Dorigo, "Implicit Parallelism in Genetic Algorithms", *Artificial Intelligence*, 1993, No. 61, pp. 307-314.

[4] Fitzpatrick, J. M., y J. Grefensette, "Genetic Algorithms in Noisy Environments", *Machine Learning*, No. 3, 1988, pp. 101-120.

[5] Forrest, S., y M. Mitchell, "Relative Building-Block Fitness and the Building Block Hypothesis", *Foundations of Genetic Algorithms* 2, 1993, pp. 109-126.

[6] Goldberg, D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989.

[7] Holland, J. H., *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, 1975.

[8] Mitchell, M., *An Introduction to Genetic Algorithms*, MIT Press, 1996.

[9] Mitchell, M., J. Holland y S. Forrest, "When Will a Genetic Algorithm Outperform Hill Climbing?", *Advances of Neural Information Processing Systems*, No. 6, Morgan Kaufmann, 1994, pp. 51-58.
http://www.santafe.edu/~mm/paper-abstracts.htm

[10] Stephens, C. y H. Waelbroeck, *Analysis of the Effective Degrees of Freedom in Genetic Algorithms*, Instituto de Ciencias Nucleares, UNAM. http://luthien.nuclecu.unam.mx:80/~nncp/genetic.html, 1996.

[11] Syswerda, G., "Uniform Crossover in Genetic Algorithms", en Schaffer, D. (editor), *Proceedins of the Third International Conference on Genetic Algorithms*, George Mason University, Morgan Kaufmann Publishers, 1989, pp. 2-9.

[12] Whitley, D., *A Genetic Algorithm Tutorial*, Technical Report CS-93-103, Computer Science Department, Colorado State University, 1993. También en *Statistics and Computing*, Vol. 4, 1994, pp. 65-85.

III. VARIACIONES SOBRE UN TEMA DE HOLLAND

EN los capítulos anteriores se presentó el algoritmo genético simple (AGS). El lector observador habrá notado que éste posee características arbitrarias muy particulares dentro de un esquema que permite, en principio, múltiples variantes. El AGS se caracteriza por utilizar un esquema de selección particular llamado selección proporcional, un esquema de cruzamiento de un solo punto de corte, mutaciones uniformemente distribuidas en la población y que opera con la misma probabilidad para todos los individuos durante todas las generaciones en que el AGS es ejecutado. También se ha presentado al lector un esquema de codificación del dominio del problema en cadenas binarias de una cierta longitud, que permanece fija al igual que el tamaño de las poblaciones que constituyen las distintas generaciones del AG.

No sólo es cierto que esto no tiene por qué ser siempre así, sino que además, en muchos casos no es conveniente que lo sea. En este capítulo se presentará al lector, a través de dos casos de estudio, diversas características que pueden complementar o reemplazar a las ya mencionadas.

III.1 CASO DE ESTUDIO: EL PROBLEMA DEL AGENTE VIAJERO

III.1.1 Codificación del dominio

III.1.1.1 Codificación binaria

La marcada preferencia por usar representaciones binarias de soluciones en los algoritmos genéticos, típicamente se deriva de la teoría del esquema, que trata de analizar a los AGs en función de su comportamiento esperado al muestrear esquemas. El argumento fundamental para justificar el fuerte énfasis en los alfabetos binarios se deriva del punto de vista de procesamiento de esquemas, por el hecho de que si se usa un alfabeto binario el número de esquemas es máximo para un número finito de puntos de búsqueda (véase [1], pp. 40-41). Consecuentemente, la teoría del esquema favorece la representación binaria de las soluciones. Pero los AGs no están restringidos a genomas codificados en binario. De hecho, existen casos considerables en que un genoma codificado en binario no es conveniente. Por ejemplo, el frecuentemente citado problema del agente viajero (TSP, por sus siglas en inglés) pertenece a este tipo. En pocas palabras, en el problema del agente viajero se trata de encontrar la trayectoria cíclica más corta

que debe seguir el agente a través de todas las ciudades de un conjunto predefinido, visitando cada ciudad sólo una vez. Las distancias entre las ciudades pueden expresarse fácilmente con una matriz de adyacencias como la que se muestra en la tabla T.III.1.A

TABLA T.III.1.A: *Matriz de distancias para el problema del agente viajero*.

	A	B	C	D	E
A	0	20.4	22.2	29.3	17.4
B	20.4	0	29.7	21.8	16.7
C	22.2	29.7	0	19.3	12.6
D	29.3	21.8	19.3	0	11.9
E	17.4	16.7	12.6	11.9	0

TABLA T.III.1.B: *Las mejores 10 rutas para el problema del agente viajero*.

Número	Ruta	Distancia
1	CEDBA	66.7
2	DECAB	67.1
3	CDEBA	68.3
4	BDECA	68.5
5	DCEBA	69.0
6	DCEAB	69.7
7	BEDCA	70.1
8	BDCEA	71.1
9	DEBAC	71.2
10	DBAEC	72.2

Primera Ciudad	Segunda Ciudad	Tercera Ciudad	Cuarta Ciudad	Quinta Ciudad
100	011	001	000	010

FIGURA F.III.1.A: *Una posible codificación para el problema del agente viajero.*

Aún en esta simple instancia, puede apreciarse la complejidad computacional inherente al problema. Hay *n!* rutas posibles (en el ejemplo n = 5, es decir 5x4x3x2 = 120 rutas). Aún si se considera que hay, de hecho, la mitad de trayectorias (debido a que toda trayectoria es equivalente a su inversa, por ejemplo ABCDE = EDCBA) por examinar, la enumeración exhaustiva sólo es posible cuando *n* es pequeña. De hecho no se conoce un algoritmo determinístico que

resuelva el problema en un tiempo que dependa polinomialmente de *n* para cualquier *n* (lo que hace del problema del agente viajero un problema NP-completo). En la tabla T.III.1.B se muestran las 10 mejores rutas para este problema.

¿Cómo puede atacarse este problema con un AG? La respuesta parece sencilla. Se procederá de la siguiente manera: *a)* codificar cada ciudad como un número binario; *b)* considerar que una solución en la concatenación de tantos números como ciudades haya; *c)* generar un conjunto de tales candidatos a solución; *d)* aplicar un AG. Un posible individuo de tal población se muestra en la figura F.III.1.A.

Surgen inmediatamente dos problemas. El primero es obvio: no puede permitirse que una ciudad, por definición, se repita en el genoma; ni puede haber números de ciudades fuera del rango especificado. En este caso, las combinaciones binarias 101, 110 y 111 están prohibidas. Por lo tanto, debe revisarse la suposición de que los elementos de la población inicial son generados aleatoriamente. Debe garantizarse un conjunto inicial de individuos correctos. El segundo problema es un poco más sutil pero se relaciona con el primero. Aún forzando a la población inicial para que cumpla con las dos restricciones anteriores, se debe ser cuidadoso al aplicar cruzamiento y mutación. Los genomas resultantes pueden inducir ciudades repetidas y/o inválidas.

¿Cómo enfrentamos estos dos problemas? Una posibilidad es aplicar un castigo a los individuos incorrectos. Con esto se espera disuadir al algoritmo de proponer los esquemas equivocados. En este caso, parecería que el AG podría gastar (y lo hace) mucho tiempo mientras aprende que las combinaciones prohibidas no son deseables.

Otra opción es aplicar un algoritmo reparador tal que la correctez es restaurada después de aplicar los operadores genéticos. En este caso no parece intuitivo que, cuando se reparan los genomas incorrectos, no se perderá parte de la información que ha incorporado al proceso genético.

Finalmente, puede dejarse la codificación binaria de lado y trabajar sobre una base diferente. Sin embargo, cuando se trabaja con una base diferente de 2 se cae en un tipo diferente de problemas. Por ejemplo, ¿cómo puede entenderse el cruzamiento? ¿Cómo se muta?

III.1.1.2 Codificación no binaria

Quizás la manera más natural de representar a los individuos de una población de posibles soluciones para el problema del agente viajero sea asignar un identificador (un carácter o un número entero) a cada ciudad, establecer que en cada posición de la cadena genética que constituye un individuo debe colocarse uno de estos identificadores, es decir, cada alelo es un identificador (hasta ahora en los AGs presentados cada alelo podía ser sólo *0* ó *1*). Se establecen las restricciones de que en un individuo debe aparecer cada identificador de ciudad una y

sólo una vez, y que todos los individuos tienen el mismo primer alelo (es decir todos los ciclos representados en una población comienzan en la misma ciudad).

En estas condiciones es sencillo asegurar que la población inicial sólo contendrá cadenas válidas. Supóngase que *n* denota el número de ciudades.

Para *i* desde 1 hasta *N* (tamaño de la población)
  Hacer una lista *L* de los *n* identificadores de ciudad distintos
  Colocar el primer identificador en la primera posición del código genético del *i*-ésimo individuo (primer alelo)
  Eliminar el primer identificador de la lista *L*

Para cada posición *j* del código genético del individuo, desde 2 hasta *n*
  Elegir aleatoriamente un identificador de *L*
  Colocar ese identificador en la posición *j* del código del individuo *i*
  Eliminar el identificador de *L*

Este algoritmo proporciona una población inicial de N cadenas válidas para el problema del agente viajero. Todas ellas denotan un ciclo posible, no se repiten ni faltan ciudades y todos los ciclos comienzan en la misma ciudad. En este caso la representación no es binaria, aun cuando, finalmente, en la memoria de la computadora todos los identificadores de ciudad serán almacenados en binario, para el AG cada identificador constituye un alelo, un ente indivisible que no puede ser partido ni alterado internamente, el análogo de cada bit en nuestras representaciones anteriores. En la figura F.III.1.B se muestra un conjunto de posibles individuos de una población.

Ahora se debe asegurar que los operadores genéticos mapeen individuos válidos en individuos válidos.

III.1.2 Operadores

Supóngase que se utiliza el esquema de codificación no binaria descrito antes. Si bajo este esquema de codificación se utilizara el cruzamiento de un punto, comenzarían a surgir individuos no válidos en la población. Supóngase que se tienen los individuos válidos: ACDBE y ADECB, ahora supóngase que, utilizando cruzamiento de un punto, se decide cortar ambas cadenas entre los alelos 3 y 4, se generarían entonces las cadenas descendientes: ACDCB y ADEBE, ambas inválidas dado que repiten ciudades. ¿Qué se puede hacer en este caso? Hay cuatro opciones. Una es la ya mencionada de castigar a cada cadena inválida haciendo que tenga una calificación muy baja por violar las reglas; otra es la de corregir cada cadena inválida mapeándola a una válida; se puede también cambiar el mecanismo de codificación de tal forma que los operadores genéticos conocidos no violen las reglas y, la otra, más barata computacionalmente, es

diseñar operadores genéticos de cruza y mutación que generen siempre cadenas válidas.

A	B	D	C	E	Individuo 1
A	D	E	C	B	Individuo 2
A	E	B	D	C	Individuo 3

FIGURA F.III.1.B: *Tres posibles individuos codificados en caracteres para el problema del agente viajero con cinco ciudades*

III.1.2.1 Operador de cruza

Es posible diseñar operadores genéticos de cruza y mutación que siempre generen cadenas válidas. Un ejemplo de operador de cruza que hace esto es el cruzamiento uniforme ordenado.

Supóngase que se poseen dos cadenas genéticas P1 y P2, es decir, dos individuos de la población que denotan dos ciclos diferentes en la gráfica de las ciudades. Una manera de generar dos individuos nuevos a partir de estos dos es la siguiente: Sea M una cadena binaria de longitud en bits igual a la longitud de los individuos, es decir, el número de ciudades. Por cada bit de M se lanza una moneda al aire y si cae cara se pone un *1* en la posición correspondiente de M, en otro caso se pone un *0*. Este proceso genera una máscara M con, aproximadamente, el mismo número de 1´s que de 0´s. Sea H una cadena genética vacía, se copia en H las posiciones de P1 que corresponden a 1´s en M. Para que H sea una cadena válida debe incluir aquellas ciudades de P1 que corresponden a 0´s en M, así que se procede ahora a colocarlas en H, pero en el orden en que aparezcan en P2. Así generamos a H que visita al mismo tiempo algunas ciudades que su padre P1, y el resto son visitadas en el mismo orden que en su padre P2.

III.1.2.2 Cruzamiento uniforme ordenado

1. Se seleccionan dos padres P1 y P2 de la población. Sea *l* la longitud de dichos padres (i.e., el número de ciudades).
2. Se genera una máscara M con longitud de *l* bits, cada bit de la cadena se genera aleatoriamente, 0 y 1 son equiprobables.
3. Sea F una cadena genética vacía y *faltan* = 1
4. Por cada posición *i* de una cadena genética H (descendiente de P1 y P2) inicialmente vacía:
   Si M[*i*] = 1 entonces
   H[*i*] = P1[*i*]
   si no

F[faltan] = P1[*i*]
faltan = faltan + 1

5. Permutar F de tal forma que aparezcan las ciudades en el mismo orden en el que aparecen en P2
6. Sea $k = 1$
7. Por cada posición *j* vacía de H
   H[*j*] = F[*k*]
   $k = k + 1$

En la figura F.III.1.C se ilustra un ejemplo de este tipo de cruzamiento.

A	B	D	C	E	Individuo 1
A	D	E	C	B	Individuo 2
1	0	1	1	0	Máscara
A		D	C		Hijo 1 (se copia de ind. 1, faltan B y E)
A		E	C		Hijo 2 (se copia de ind. 2, faltan D y B)

A	B	D	C	E	Individuo 1
A	D	E	C	B	Individuo 2
1	0	1	1	0	Máscara
A	E	D	C	B	Hijo 1 (E aparece antes que B en ind. 2)
A	B	E	C	D	Hijo 2 (B aparece antes que D en ind. 1)

FIGURA F.III.1.C: *Los dos pasos para generar un par de descendientes de una pareja de individuos con cruzamiento uniforme ordenado.*

III.1.2.3 Mutación

Si se utiliza el operador de mutación tradicional en el problema del agente viajero con la codificación ya descrita, al igual que en el caso del cruzamiento, se pueden generar cadenas inválidas. Pero también es posible definir un nuevo operador de mutación que asegure que siempre que altere una cadena válida el resultado será otra cadena válida. Se procederá de manera similar al caso del cruzamiento.

1. Sea P1 una cadena genética de longitud *l*
2. Sea M una cadena binaria de longitud *l* y p la probabilidad de mutación
3. Para cada posición *i* de M desde 2 hasta *l* (se excluye la primera posición)

Se elige aleatoriamente el valor de M[*i*], con probabilidad *p* se elige 1, con probabilidad 1-*p* se elige 0

5. Si el número M tiene exactamente un 1 en la posición *r*, entonces se elige aleatoriamente otra posición de M distinta de 1 y de *r* para colocar otro 1
6. Sea Temp una cadena genética inicialmente vacía y cont = 1
7. Por cada posición *j* de P1 desde 2 hasta *l*
   Si M[*j*] = 1 entonces
   Temp[cont] = P1[*j*]
   cont = cont + 1
   P1[*j*] = vacío
8. Permutar los elementos de Temp de tal forma que ninguno ocupe su posición original en Temp
9. *k* = 1
10. Para cada *i* desde 1 hasta *l*
    Si P1[*i*] = vacío entonces
    P1[*i*] = Temp[*k*]
    k = *k* + 1

Con este algoritmo se asegura que las cadenas generadas por mutación a partir de cadenas válidas son también válidas, dado que únicamente se altera el orden en que son visitadas las ciudades, aunque cabe aclarar que, dado lo que se especifica en el paso 5, la probabilidad de mutación ya no es *p*.

III.1.3 La función de adaptación

En los problemas planteados en los capítulos anteriores se tenía una función de evaluación única e invariable para todas las generaciones del AG. Pero en el caso del agente viajero, ¿cuál debe ser la función de evaluación de cada individuo?

Se pretende minimizar el costo de un viaje redondo que pase por todas las ciudades. Desde aquí comienzan los problemas. Hasta ahora, en el texto siempre se había tratado con problemas de maximización que están naturalmente adaptados para los AGs. Es bien sabido que todo problema de minimización puede ser traducido a uno de maximización simplemente multiplicando por –1 la función de evaluación. Así, al maximizar ésta, se está minimizando implícitamente la función objetivo. Pero en el caso del agente viajero, dado que todos los costos son no negativos, al multiplicar por –1 se obtendrán valores negativos no permitidos por el esquema de selección proporcional. Una alternativa podría ser que el valor de adaptación de cada individuo sea la diferencia entre su costo total y el costo máximo posible de un ciclo en la instancia del problema que se pretende resolver. Pero esto significa que se debe conocer al costo máximo de un ciclo en

dicho problema. Esto es tan difícil como conocer el costo del ciclo mínimo, así que en este caso no es útil la propuesta.

Una opción útil en estos casos es cambiar el esquema de selección por uno en el que la bondad de un individuo no sea proporcional al valor obtenido mediante la función de evaluación. Un esquema que se puede usar en este caso es, por ejemplo, la selección presentada en el ejercicio 4 del capítulo 1, conocida en inglés como *linear ranking*. En este esquema, la probabilidad de selección de un individuo está dada por su posición (*ranking*) respecto al total de la población ordenada. Para aplicar este esquema, se debe ordenar la población de manera creciente en función de la longitud del ciclo que representa cada individuo, y luego se seleccionan los individuos con mayor probabilidad cuanto más cercanos aparezcan al principio de la lista.

Otra opción es utilizar un remapeo de la función objetivo (costo de viaje) para obtener una función de adaptación. Un remapeo útil para el agente viajero es el siguiente: sean f_{max} , f_{min} y f_i el costo máximo, mínimo y el del *i*-ésimo individuo en alguna población, respectivamente, la función de adaptación evaluada en el *i*-ésimo individuo es:

$$Adap(\, i\,) = (f_{max} + f_{min}) - f_i \qquad \text{(III.1.A)}$$

Haciendo esto ya es posible utilizar el conocido esquema de selección proporcional.

III.1.4 Síntesis

Nuestro análisis previo acerca del problema del agente viajero puede sintetizarse como sigue:

1. Dependiendo del problema es posible elegir una codificación binaria para el dominio de búsqueda o no.
2. Se puede optar por un conjunto de operadores genéticos que aseguren que no se pueden generar cadenas inválidas o bien,
3. Un esquema que permita disminuir la probabilidad de aparición de cadenas inválidas castigándolas cuando aparecen y forzando al AG a aprender a no generarlas, u
4. Ocasionalmente se puede decidir cambiar el esquema de codificación del dominio para poder utilizar operadores más sencillos sin que se corra el riesgo de generar cadenas inválidas.
5. Es posible remapear la función objetivo para convertirla en una función de adaptación útil, o bien
6. Cambiar el esquema de selección por alguno que no requiera cambiar la función de adaptación

A pesar de que nuestro análisis se ha centrado en el problema del agente viajero, las conclusiones son aplicables a una amplia gama de problemas en los que pueden presentarse todas o algunas de las condiciones analizadas.

III.2 CASO DE ESTUDIO: OPTIMIZACIÓN DE UNA FUNCIÓN

El siguiente caso que se estudiará, con el objetivo de explorar algunas otras variantes que pueden ser introducidas a partir del algoritmo genético simple, es el de optimización de una función. En este caso la regla de correspondencia de dicha función es:

$$z = f(x,y) = -(x-5)^2 - (y-3)^2 \qquad \text{(III.2.A)}$$

la que ya fue utilizada en el capítulo 1.

El código genético de cada individuo estará constituido entonces por un vector con dos entradas, una para la variable *x* y otra para *y*. Cada una de estas entradas es un número que será codificado en binario. Serán consideradas varias alternativas para hacer esto y se procederá luego a comparar los resultados de cada una de estas posibilidades.

III.2.1 Codificación del dominio

III.2.1.1 Codificación en punto fijo

Se tienen varias opciones para codificar cada una de las entradas del vector (*x*,*y*) en cada individuo de una población. Una de ellas es la de utilizar el formato de punto fijo. Para hacer esto el primer paso es determinar, *a priori*, cuántos bits se utilizarán para codificar tanto la parte entera como la fraccionaria, de una entrada cualquiera en el vector. Este no es un problema trivial. En general, no se sabe en qué rango debe encontrarse el punto en el que la función tiene su máximo, así que al decidir *a priori* el conjunto de valores representables puede ocurrir que ese punto se encuentre fuera de dicho conjunto y por lo tanto sea imposible hallarlo bajo el esquema elegido. Por desgracia, a menos que se sepa algo acerca del problema que permita restringir el dominio de búsqueda y por tanto elegir una codificación que lo cubra de la mejor manera posible, este problema no tiene solución.

En nuestro caso se ha elegido tener 3 bits para representar la parte entera de un número (entrada del vector) y 24 bits para representar la parte fraccionaria. Además, se ha añadido un bit adicional para denotar el signo del número: 0 si es no negativo y 1 si es negativo. El esquema utilizado para representar a cada posible solución (individuo) se muestra en la figura F.III.2.A.

Dentro de este mismo esquema de representación de punto fijo puede optarse también por diversos métodos de representación de la parte entera y la fraccionaria. En este caso se utilizarán dos diferentes: la representación en notación binaria pesada común y la codificación de Gray.

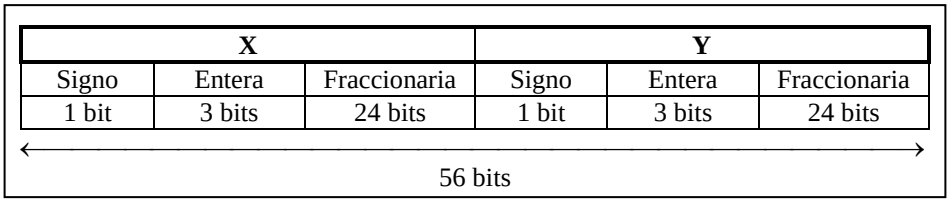


FIGURA F.III.2.A: *Genoma en formato de punto fijo.*

III.2.1.2 Codificación binaria pesada

En la vida cotidiana normalmente se utiliza el sistema numérico posicional en base 10, en el que el número 1235.2 representa un millar, dos centenas, tres decenas, cinco unidades y dos décimos. Es decir:

$$1235.2=1x10^3 + 2x10^2 + 3x10^1 + 5x10^0 + 2x10^{-1}$$

Este esquema es generalizable. Se puede hablar entonces de un sistema numérico posicional en una base arbitraria B y escribir números en este sistema como una cadena de dígitos: $a_k \ldots a_0 \, . \, a_{-1} \ldots a_j$, donde cada a_i es un dígito cuyo valor $|a_i| \in \{0, ..., B\text{-}1\}$ y por lo tanto el valor del número es:

$$a_k \times B^k + \ldots + a_0 \times B^0 + a_{-1} \times B^{-1} + \ldots + a_j \times B^{-j}$$

El sistema numérico que aquí se ha llamado binario posicional o binario pesado es un caso particular de sistema numérico posicional cuya base es 2. Así cada dígito de un número binario en este sistema es, claramente 0 ó 1, y el valor de cada dígito dentro de un número es el múltiplo de la potencia de dos que le corresponde por su posición dentro del mismo. Por ejemplo, el número binario 10010110.01 en este sistema tiene el valor:

$$1\times2^7 + 1\times2^4 + 1\times2^2 + 1\times2^1 + 1\times2^{-2} = 150.25$$

III.2.1.3 Codificación binaria de Gray

El esquema de codificación de Gray no es un sistema posicional y se utiliza en algoritmos genéticos por una característica peculiar. Ocurre que cualesquiera dos números consecutivos tienen un código de Gray que difiere en un solo bit, cosa que no ocurre con el esquema de codificación en binario pesado donde, por ejemplo, el 3 (011) difiere del 4 (100) en tres bits. Esto significa que en Gray los códigos de elementos del dominio que están cerca también están cerca, es decir, fenotipos cercanos se mapean en genotipos cercanos y viceversa.

Para obtener el código Gray (g_k, ..., g_0), de un número entero *r* entre 0 y 2^{k+1}-1 (incluyendo los extremos), cuya expresión en binario pesado es (b_k, ..., b_0) se hace lo siguiente:

1. $g_k = b_k$
2. Para i desde k-1 hasta 0
3. $g_i = b_{i+1}$ XOR b_i

Donde la operación XOR es la disyunción exclusiva entre bits. Esto es, resulta 1 cuando los bits sobre los que opera son diferentes y resulta 0 cuando ambos son iguales. En la tabla T.III.2.A aparecen los códigos de Gray y binario pesado para los primeros 16 enteros expresables en 4 bits.

III.2.1.4 Codificación en punto flotante

El estándar más usual para la representación de números en punto flotante es el de la IEEE (*Institute of Electrical and Electronics Engineers*). Un número real en punto flotante con precisión sencilla se representa como una palabra de 4 bytes. El bit más a la izquierda del primer byte (B1) representa el signo del número; los bits del 6 al 0 de B1 *y* el bit más a la izquierda del byte 2 son el exponente del número. El exponente está representado en base 2 y se expresa en exceso 127, esto es, el exponente actual se obtiene restando 127 del número binario pesado convencional. Los 23 (32-9) bits restantes representan la mantisa. La mantisa está justificada a la izquierda y se supone que tiene un punto antes del número binario. Adicionalmente, se agrega al número un *1* implícito. El número hexadecimal 3F400000 en este formato representa al número decimal +0.75, mientras que el número hexadecimal C1200000 representa al número decimal –10. Un genoma hipotético para el problema que nos ocupa puede ser como el representado en la figura F.III.2.B.

De manera alternativa, los números *x* y *y* pueden expresarse en el formato de doble precisión de IEEE para punto flotante. La representación de doble precisión es una extensión natural de la representación simple, de acuerdo con lo siguiente:

El número se representa con 8 bytes, en vez de 4.
El exponente está representado por los 11 bits más a la izquierda (después del bit de signo). Ahora se expresa en exceso 1023.
La mantisa está formada por los 52 bits más a la derecha.

De acuerdo con lo anterior, la mantisa más grande sin signo que puede representarse es $2 - 2^{-52}$, para un rango numérico ρ de $2^{-1023} \leq |\rho| \leq (2 - 2^{-52})\ 5\ 2^{+1024}$ o, aproximadamente $2^{-1023} \leq |\rho| \leq 2^{+1025}$.

Un genoma para el mismo problema puede ahora ser expresado como se muestra en la figura F.III.2.C.

TABLA T.III.2.A: *Códigos de Gray y binario pesado convencional para los primeros 16 números naturales.*

Gray	Binario pesado	Número
0000	0000	0
0001	0001	1
0011	0010	2
0010	0011	3
0110	0100	4
0111	0101	5
0101	0110	6
0100	0111	7
1100	1000	8
1101	1001	9
1111	1010	10
1110	1011	11
1010	1100	12
1011	1101	13
1001	1110	14
1000	1111	15

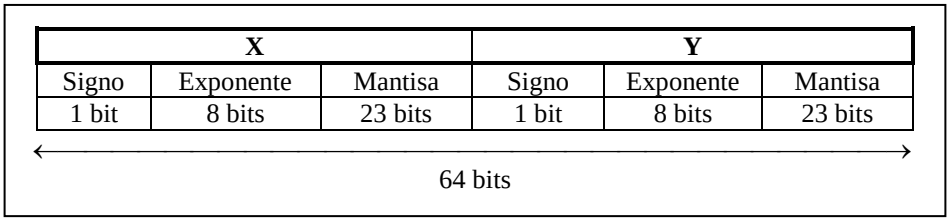


FIGURA F.III.2.B: *Genoma en formato de punto flotante con precisión sencilla.*

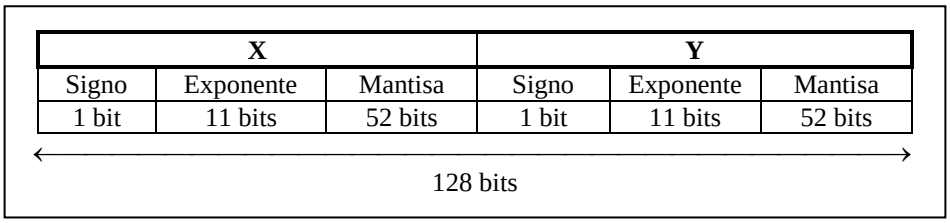


FIGURA F.III.2.C: *Genoma en formato de precisión doble*

III.2.1.5 ¿Cuál es la mejor opción?

Cabe ahora preguntarse: ¿Cuál de estas opciones es la mejor? ¿Influirá de alguna manera el esquema de codificación utilizado en el comportamiento del algoritmo genético? ¿Se debe esperar que el algoritmo genético funcione bien sin importar el esquema utilizado?

Todo esto es especialmente interesante cuando se considera el cruzamiento de un punto. Con un formato de punto fijo se están explorando en los AGs diferentes alternativas, en las que se ocurre alguno de estos casos.

a) El punto de cruzamiento cae a la derecha del número Y; el segmento por intercambiar pertenece a Y.
b) El punto de cruzamiento está exactamente donde comienza Y; el segmento por intercambiar *es* el número Y.
c) El punto de cruzamiento es tal que un segmento de X y todo Y están incluidos en el segmento por intercambiar.

En el caso (a), por lo tanto, se comienza con dos genomas que representan a los número X_1 y Y_1 para el individuo 1, y los números X_2 y Y_2 para el individuo 2. Después del cruzamiento, los dos individuos involucrados se transforman en X_1 y Y_1', por un lado y en X_2 y Y_2' por el otro.

En el caso (b), después del cruzamiento los dos individuos involucrados se transforman en X_1 y Y_2 y X_2 y Y_1, respectivamente.

Finalmente, en el caso (c), después del cruzamiento los dos individuos son transformados X_1' y Y_2 y X_2' y Y_1, respectivamente.

Con un formato de punto flotante se están explorando en los AGs alternativas más complejas que caen dentro de los siguientes casos.

a) El punto de cruza está a la derecha de la mantisa del número Y; el segmento por intercambiar es un segmento de la mantisa de Y.
b) El punto de cruza está exactamente donde empieza la mantisa de Y; el segmento por intercambiar es la mantisa de Y.
c) El punto de cruza es tal que un segmento del exponente de Y y toda la mantisa de Y forman el segmento por intercambiar.

d) El punto de cruza está exactamente donde empieza la mantisa de Y; el segmento por intercambiar está formado por el exponente de Y y su mantisa.
e) El punto de cruza incluye (¿exactamente?) todo Y.
f) El punto de cruza incluye un segmento de la mantisa de X y a Y.
g) El punto de cruza incluye la mantisa de X y a Y.
h) El punto de cruza incluye un segmento del exponente de X, su mantisa y a Y.
i) El punto de cruza incluye el exponente de X, su mantisa y a Y.

X_1			**Y_1**		
Signo	**Exponente**	**Mantisa**	**Signo**	**Exponente**	**Mantisa**

X_2			**Y_2**		
Signo	Exponente	Mantisa	Signo	Exponente	Mantisa

FIGURA F.III.2.D: *Individuos antes de la cruza.*

X_1				**Y_1**		
Signo	**Exponente**	**Mantisa (Alto)**	Mantisa (Bajo)	Signo	Exponente	Mantisa

X_1				**Y_1**		
Signo	Exponente	Mantisa (Alto)	**Mantisa (Bajo)**	**Signo**	**Exponente**	**Mantisa**

FIGURA F.III.2.E: *Individuos después de la cruza.*

¿Puede esperase que tal esquema permita al AG explorar eficientemente el dominio de búsqueda? Para ilustrar los problemas que enfrentará el AG, considérese el caso siguiente.

Caso (f). Se empieza con dos genomas: $X_1 + Y_1$ y $X_2 + Y_2$. Se supondrá que estos dos conjuntos de números son razonablemente adecuados en el sentido de que no están demasiado lejos del óptimo deseado. La figura F.III.2.D muestra a los individuos antes del cruzamiento.

Supóngase que se lleva a cabo el cruzamiento. Esto se ilustra en la figura F.III.2.E.

Ahora supóngase que $\vec{v}_1$=(5.4589, 3.1133) y $\vec{v}_2$=(4.6875, 3.2188). La representación hexadecimal de estos vectores (en formato de punto flotante) es

$\vec{v}_1$=(49AEAF4F, 4047404E) y $\vec{v}_2$=(4095FFFF, 404E00D1). Los genomas G_1 y G_2 son, entonces, 40AEAF4F4047404E y 4095FFFF404E00D1. Supóngase que el punto de cruza es el bit 13^{avo}. Entonces, después de la cruza se tendrán los vectores resultantes $\vec{v}_1$=(40A5FFFF, 404E000D1) y $\vec{v}_2$=(409EAF4F, 4047404E). En decimal, estos vectores son $\vec{v}_1$=(5.1875, 3.2188) y $\vec{v}_2$=(4.9589, 3.1133). Es importante notar que en formato de punto flotante, el rango numérico explorado por el AG es, en principio, mucho mayor que el explorado con la representación de punto fijo. Por ejemplo, si un genoma de punto fijo hipotético está formado por 10 bits en la parte entera y 21 bits en la parte fraccionaria, el valor absoluto más grande representable es más pequeño que 2^{11} (el genoma tiene 32 bits). En contraste, una variable en punto flotante de precisión sencilla puede manejar un valor absoluto de hasta 2^{128} (para la misma longitud de 32 bits).

Así que, ¿se puede esperar que el algoritmo genético capture la esencia del problema? ¿Será capaz de identificar los esquemas adecuados?

Michalewicz ([2], pp. 105 y 106) ha reportado una serie de estudios que arrojan las siguientes conclusiones:

1. La representación en punto flotante hace la ejecución del AG más rápida y permite una mayor precisión.
2. Es más fácil diseñar operadores especiales, que incorporen cierto conocimiento del problema, para la representación en punto flotante.
3. En problemas cuya solución debe ser muy precisa, la representación en punto flotante es mejor opción, debido a que la de punto fijo involucraría genomas prohibitivamente grandes.
4. La representación en punto flotante evita lo que se ha dado en llamar acantilados de Hamming, que consisten en que, una vez que el algoritmo ha descubierto que la función tiene su máximo en los valores positivos de una variable, por ejemplo, una mutación en el bit de signo de una instancia de dicha variable, ocasionaría que se explorara nuevamente el subespacio de valores negativos. Es decir, el hecho de tratar a la cadena de bits de punto fijo como un número sin signo, dado que el operador de mutación no hace consideraciones especiales con ningún bit, puede ocasionar saltos indeseables muy grandes, aun cuando el genoma mutado difiera del original en tan sólo un bit.

III.2.2 Tres algoritmos

Seguramente, si el lector ya realizó los ejercicios de programación de los capítulos previos, se habrá dado cuenta de que en ocasiones el AG encuentra, en alguna generación, una buena aproximación a la solución del problema que se le ha planteado y que en la siguiente generación esta buena propuesta desaparece, casi

con seguridad el promedio de la calificación de la población mejoró. Pero esa buena solución que se había encontrado ya no está en la población.

Ocurre que en el AGS no existe ningún medio para asegurar que esa buena aproximación que ya se encontró se preserve. Puede ocurrir que ni siquiera fuera seleccionada para dejar descendientes en la siguiente generación. Es cierto que el mejor individuo de la población tiene la más alta probabilidad de ser seleccionado por la ruleta, pero hay una probabilidad distinta de cero de que esto no ocurra. También puede ser que sea elegido y se pierda al cruzarse con algún otro seleccionado distinto de él mismo, o bien que logre sobrevivir a la cruza pero la mutación lo altere. En fin, no se provee de un mecanismo que asegure que el mejor individuo de la generación *i* pase intacto a la generación *i+1*.

Esto trae como consecuencia que la mejor calificación de cada generación del AGS pueda fluctuar. Sería deseable que el comportamiento de la calificación máxima de la población fuera monótono. En específico no decreciente. Asegurar que el mejor individuo de la generación *i+1* tendrá una calificación al menos tan alta como el mejor de la generación *i*.

Para lograr esto se inventó el *elitismo*. En términos generales este mecanismo consiste en conservar, en la generación *i* de un AG, a los *k* mejores individuos de las últimas *r* generaciones. En su variante más simple consiste en conservar (copiar sin alteraciones) al mejor individuo de la generación inmediata anterior. Esto asegura el comportamiento monótono de la mejor calificación por generación y más adelante (capítulo 4) se mostrará que también es condición necesaria y suficiente para la convergencia de un AG. En lo sucesivo se denotará como AGT el algoritmo genético simple al que se le ha añadido esta característica, es decir: AGS + elitismo = AGT.

El AGT es una variante sencilla del AGS, sin embargo, puede pensarse en variantes más complicadas. Una de éstas, que será analizada con más detalle en el capítulo siguiente, es la que se ha denominado algoritmo genético ecléctico (AGE en lo sucesivo). En este algoritmo no convencional se incluyen las siguientes características: estrategia de selección determinística, elitismo, autoadaptación y la invocación aleatoria (siguiendo cierta estrategia) de un escalador de ascenso por mutación aleatoria.

Como se ha dicho, el AGE será analizado con detalle en el capítulo siguiente, por ahora será suficiente saber que:

1. La selección determinística se refiere a que no es un proceso aleatorio de selección sino que, para cada individuo de la población está determinado, en función de su posición dentro de la misma de acuerdo con su adaptación, el individuo con el que deberá cruzarse.
2. La autoadaptación significa que ciertos parámetros de control del algoritmo, tales como la probabilidad de mutación y de cruza y algunos otros, son añadidos al genotipo de cada individuo en vez de ser dados *a priori* por el usuario del algoritmo. En la población inicial los valores de estos parámetros pa-

ra cada individuo son elegidos aleatoriamente (en ciertos rangos) y se les somete, al igual que el resto del genoma, a los operadores genéticos. Para determinar el valor de estos parámetros que han de usarse efectivamente para cruzar y mutar a la población, se obtiene el promedio de los valores de dichos parámetros en los individuos.

3. En el capítulo 2 se presentaron los experimentos de Mitchell con las funciones del tipo caminos regios, donde se vio que el escalador por mutación aleatoria se desempeñaba eficientemente (de hecho más que un AG) en la solución del problema. El AGE ocasionalmente corre un escalador de este tipo en vez de aplicar el proceso genético normal. Esto lo hace, por ejemplo, cada vez que en la población todos los individuos son iguales. En general, en el AGE existe un parámetro (también codificado en los individuos) que indica el número mínimo de veces que ha de ser invocado el escalador en cada generación.

III.2.3 Experimentos

Se tienen ahora tres diferentes algoritmos genéticos (AGS, AGT, AGE) y cuatro diferentes esquemas de codificación, a saber:

1. Punto fijo, codificación en binario pesado.
2. Punto fijo, codificación en Gray.
3. Punto flotante, libre (i.e., cualquier número en punto flotante es válido).
4. Punto flotante, acotado. (Se fijan *a priori*, ciertas cotas para el dominio de búsqueda. Cada vez que se genera un nuevo individuo se verifica que esté dentro de las cotas establecidas, en caso contrario se corrige.)

En el punto 4 se está suponiendo que se sabe algo acerca del problema que permite acotar el espacio de soluciones posibles.

Para los experimentos con el esquema de codificación de punto fijo se utilizó el genoma ya mencionado, representado en la figura F.III.2.A, en el que cada entrada del vector está formada por 28 bits, un 1 para el signo, tres para la parte entera y 24 para la fraccionaria. Con estas características el rango de valores representables para cada variable está entre $-(8\text{-}2^{-24})$ y $+(8\text{-}2^{-24})$. Se utilizaron, como ya se dijo, la codificación en binario pesado y en Gray.

Para los experimentos con representación de punto flotante libre y acotada se utilizó un número de precisión sencilla para cada entrada del vector, tal como se muestra en la figura F.III.2.B. En este caso, por supuesto, el panorama de las soluciones exploradas por el AG es mucho mayor que en el caso anterior. Contrariamente a la representación en punto fijo, en este caso pueden tomarse valores aproximadamente entre 2^{-127} y 2^{+129}.

¿Es deseable un rango mucho más grande? Si no se sabe nada acerca de la solución hipotética, esto parece ideal. Sin un costo excesivo (en términos de la longitud del genoma) se podrá explorar un panorama muy grande. Sin embargo, en este caso se establecieron, para el caso de la representación en punto flotante acotada, cotas para los posibles valores de *x* y *y*, lo que reduce significativamente el tamaño del dominio de búsqueda. Así, cada vez que se genera un individuo con una entrada que rebasa las cotas establecidas, esta entrada se modifica de modo que se ajuste a las restricciones impuestas y no se pierde tiempo evaluando individuos que salen del rango establecido.

Los parámetros operacionales para los algoritmos AGS y AGT se muestran en la tabla T.III.2.A. Se eligieron cuatro parejas diferentes para los valores de la probabilidad de cruza (P_c) y la probabilidad de mutación (P_m). Estos valores son consistentes con la idea de que la explotación del conocimiento obtenido por el AG es favorecido por el cruzamiento, mientras que la exploración de nuevos puntos en el dominio no debía ser demasiado destructiva.

TABLA T.III.2.A: *Parámetros para AGS/AGT.*

Número de individuos por generación	100
Número de generaciones	100
Tamaño del genoma	56
Tipo de codificación	Punto fijo, binario pesado, Gray
Semilla del generador aleatorio	1.0
(P_c, P_m)	(1,.005);(.6,.005), (.9,.005), (.9,.01)

El rango de los valores de control para el AGE se muestra en la tabla T.III.2.B. En esta tabla N_η denota la probabilidad de que sea invocado el escalador y *n* denota el número de descendientes. Dado que estos parámetros son incluidos como parte del genoma de los individuos en AGE y se autoadaptan sometidos a los operadores genéticos, los valores que aparecen en la tabla son, de hecho, utilizados para generar los valores que poseen los individuos de la población inicial y marcan también los límites dentro de los que los valores se autoadaptan.

TABLA T.III.2.B: *Rango de parámetros para AGE.*

Valor más bajo para P_c	0.7
Valor más alto para P_c	1.0
Valor más bajo para P_m	0.001
Valor más alto para P_m	0.005
Valor más bajo para *n*	1
Valor más alto para *n*	2
Valor más bajo para N_η	0.9
Valor más alto para N_η	1.0

Resultados

Los resultados de los experimentos se muestran en las gráficas F.III.2.A-F.III.2.J. En las primeras dos resulta evidente que la mejor adaptación promedio para las distintas variaciones de punto flotante es mucho peor que la obtenida para punto fijo. Esto contradice las conclusiones de Michalewicz ya mencionadas. No puede explicarse fácilmente la razón para las diferencias obtenidas. Los algoritmos son similares (al menos en el caso del AGS) y es en este caso, precisamente, cuando las diferencias se hacen más evidentes. Estos resultados son más intuitivos; un gran espacio de exploración hace la tarea de optimización más difícil.

La adaptación promedio del AGS se muestran en la figura F.III.2.C. Aquí se promediaron los cuatro conjuntos de parámetros indicados en la tabla T.III.2.A. De la gráfica, es claro que existe una diferencia importante entre las codificaciones con punto fijo, por un lado, y las codificaciones con punto flotante, por el otro. De hecho, la diferencia es tan importante que los valores para los casos de punto fijo son casi indistinguibles. Por esta razón, se han separado los datos en otras dos gráficas, que se muestran en las figuras F.III.2.D y F.III.2.E.

Algo parecido ocurrió con los resultados de AGT, mostrados en las gráficas F.III.2.F a F.III.2.H. Cabe hacer notar que, de acuerdo con las predicciones teóricas, el AGT converge al punto donde la función tiene su máximo. A diferencia de lo ocurrido con AGS, en AGT parece no haber diferencia significativa entre el esquema de codificación en binario pesado y el de Gray. También es evidente, si se comparan F.III.2.E y F.III.2.H, que el AGT exhibe características de convergencia muy superiores al AGS en la variante de punto flotante.

Si se comparan ahora las gráficas F.III.2.C, F.III.2.F y F.III.2.I se notará que el AGE posee características muy ventajosas frente a las otras variantes, converge siempre al óptimo y en un número menor de generaciones.

Finalmente, en F.III.2.J se muestra la evolución del promedio del valor de la probabilidad de mutación contenida en los individuos del AGE. Es de notarse que siempre al principio se incrementa, lo que corresponde con una fase de exploración intensa. Tanto más intensa cuanto mayor sea el tamaño del espacio de búsqueda, lo que se deduce del hecho de que es mayor en el caso de la representación en punto flotante libre. En el capítulo siguiente se analizará con más cuidado el AGE y los motivos por los que resulta ventajoso.

En los experimentos realizados, que por supuesto no esperan ser concluyentes, resulta ser que el AGE con un esquema de codificación en Gray es la opción más ventajosa.

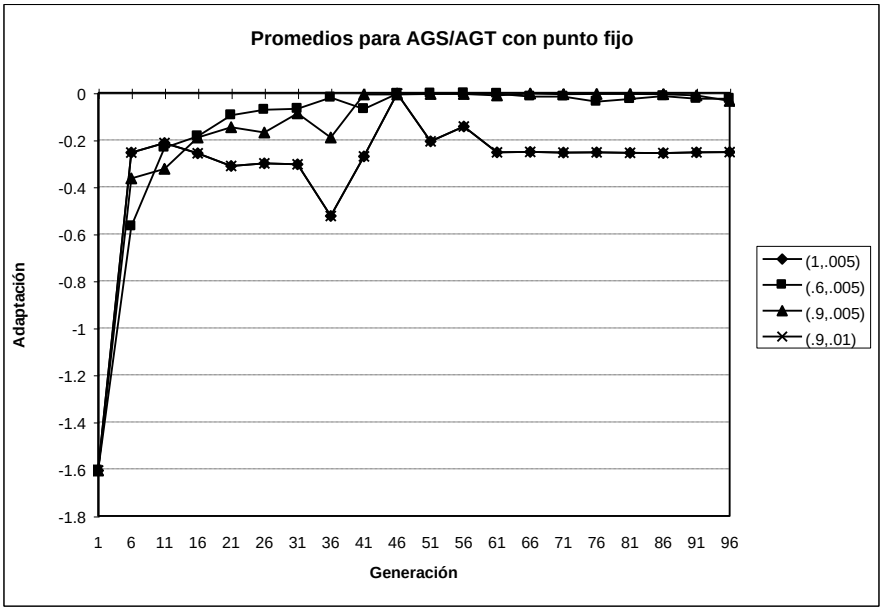


FIGURA F.III.2.A: *Promedios para representación con punto fijo en AGS/AGT.*

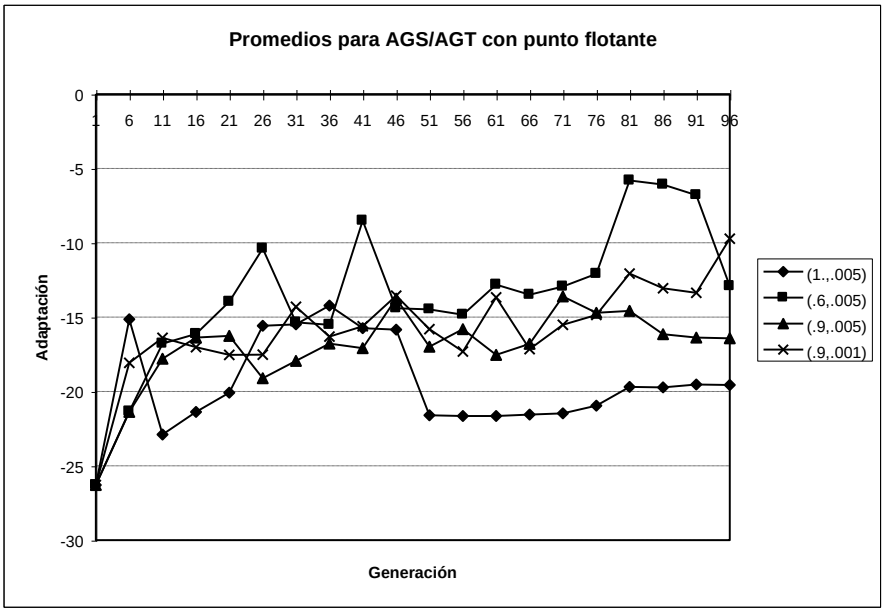


FIGURA F.III.2.B: *Promedios para representación con punto flotante en AGS/AGT.*

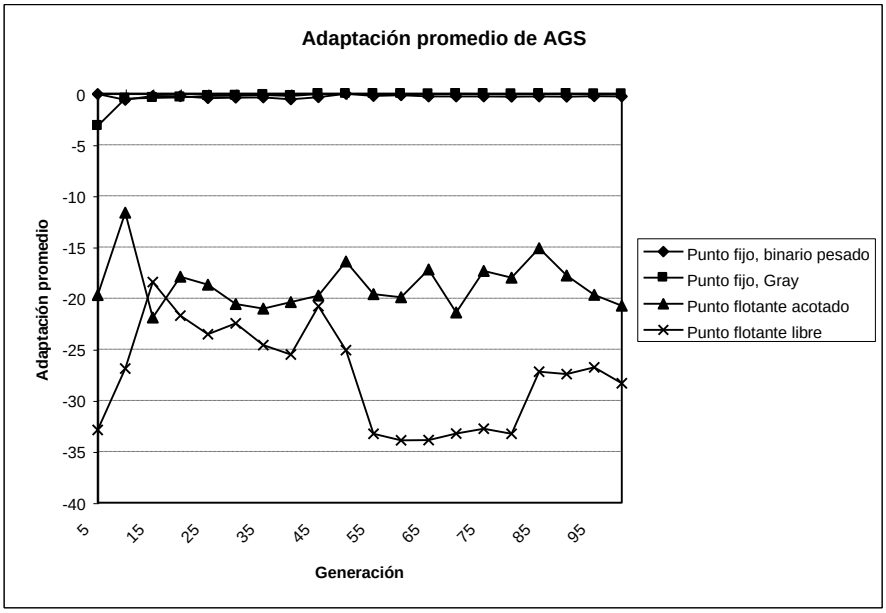


FIGURA F.III.2.C: *Adaptación promedio para AGS.*

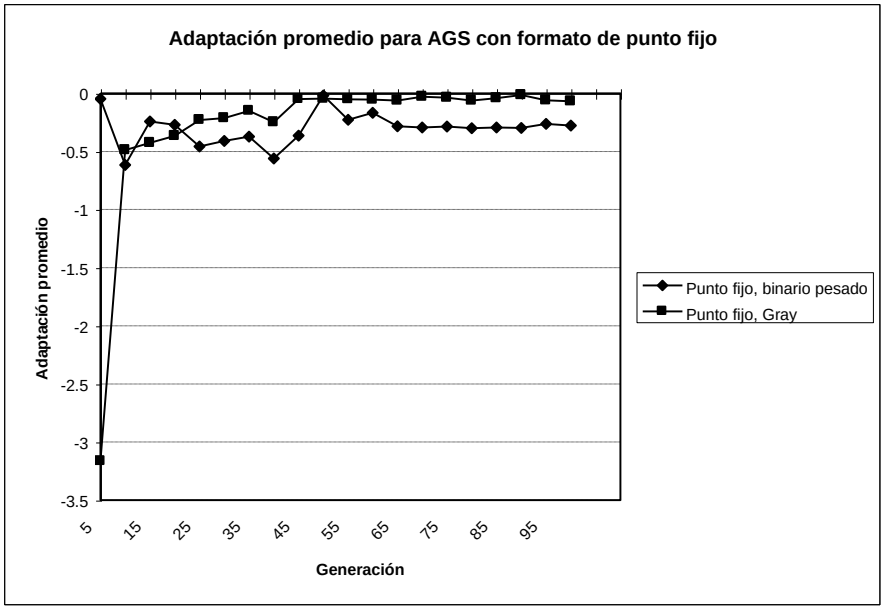


FIGURA F.III.2.D: *Adaptación promedio para AGS con punto fijo.*

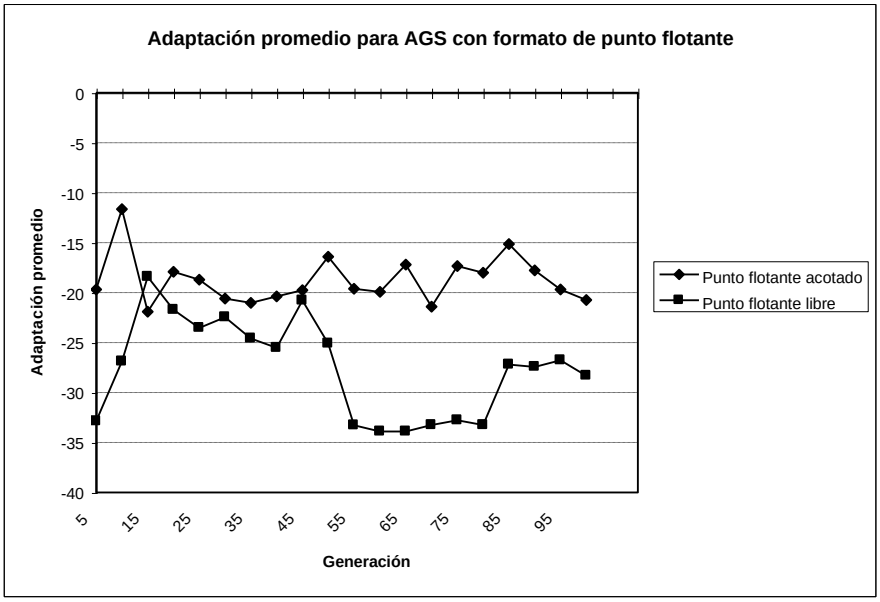


FIGURA F.III.2.E: *Adaptación promedio para AGS con punto flotante.*

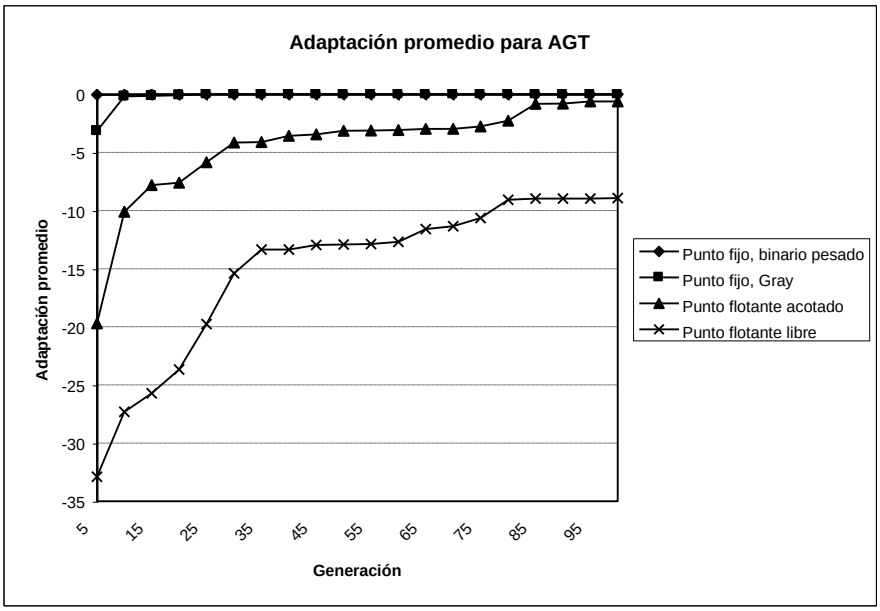


FIGURA F.III.2.F: *Adaptación promedio del AGT.*

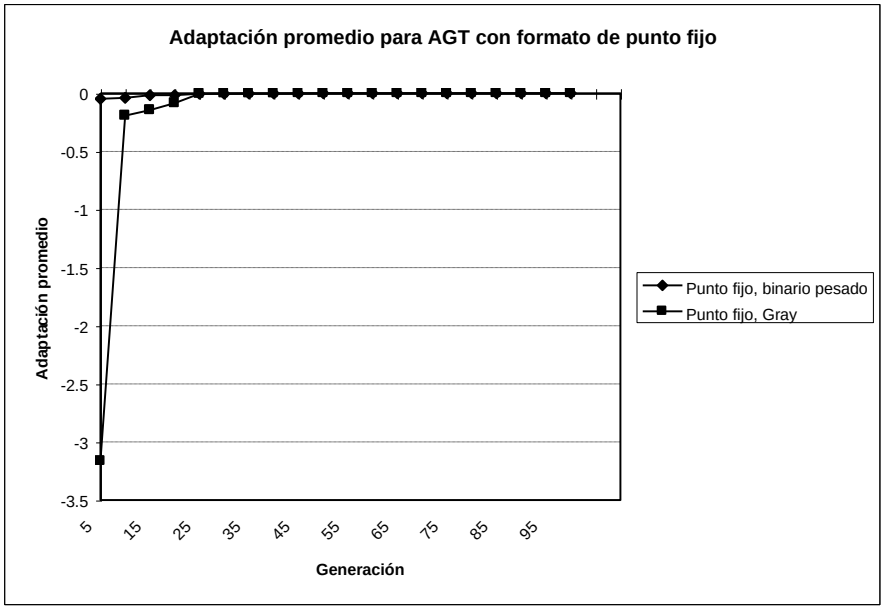


FIGURA F.III.2.G: *Adaptación promedio para AGT con punto fijo.*

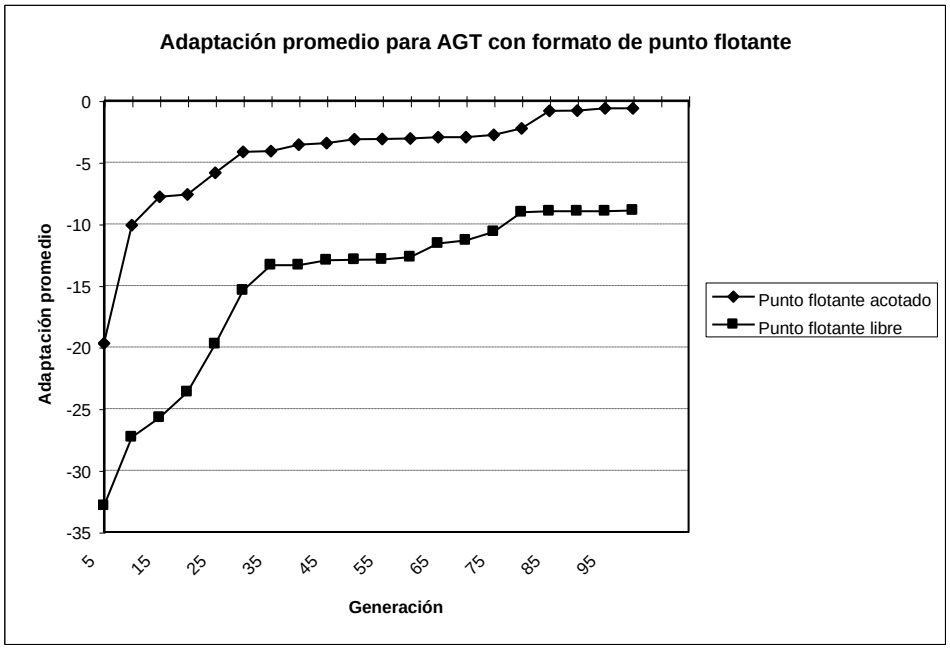


FIGURA F.III.2.H: *Adaptación promedio para AGT con punto flotante.*

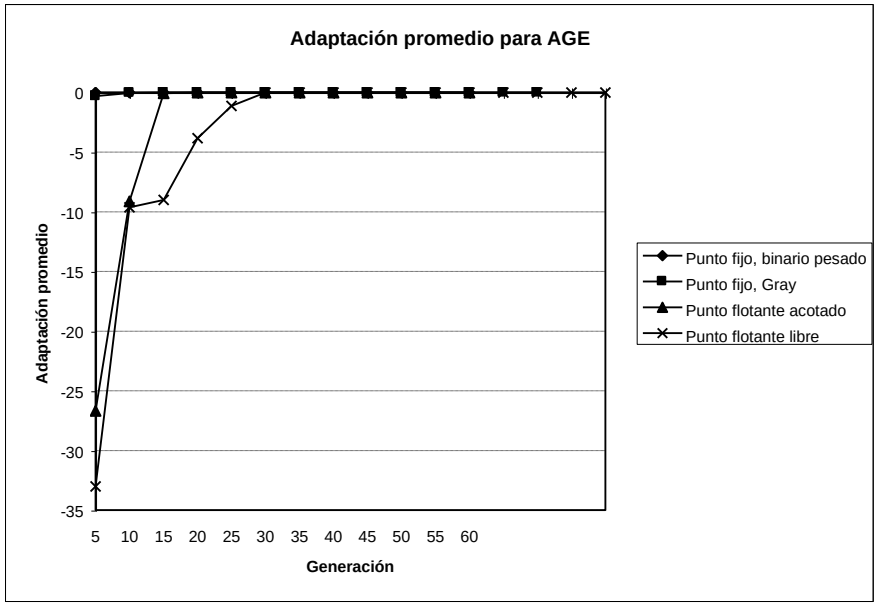


FIGURA F.III.2.I: *Adaptación promedio para AGE.*

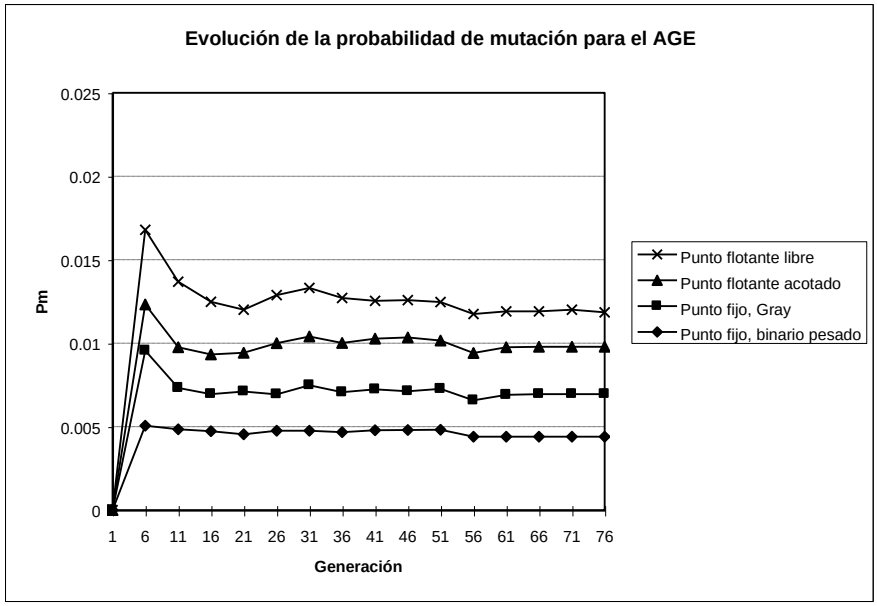


FIGURA F.III.2.J: *Evolución de la probabilidad de mutación promedio en el AGE.*

III.3 EJERCICIOS

1. Sean: a y $b=a+1$ dos enteros no negativos consecutivos, k un entero no negativo ($k \in \{0, 1, \ldots\}$) y p_m la probabilidad de mutación de cada bit en un AG. Calcule, para los esquemas de codificación en binario pesado y Gray, la distancia de Hamming y la probabilidad de que a se transforme en b mediante mutación si a es de la forma:

   a) 2^k
   b) $4k + 1$
   c) $2^k - 1$

   Suponiendo que $p_m << 1 - p_m$ ¿Qué significa esto en el contexto del AG?

2. En el formato de precisión sencilla de IEEE ¿A qué número corresponde 40490FDB?

3. Sea f un número en punto flotante de precisión sencilla en el formato de IEEE. ¿Cuál es el número ε más pequeño tal que $f + \varepsilon$ sea también representable en el mismo formato? Si se usara un formato de punto fijo, ¿cuántos bits requiere tener la parte fraccionaria para obtener la misma ε? ¿De qué manera pueden influir estos factores en el desempaño de un AG?

4. Una alternativa para representar un ciclo en el problema del agente viajero es la siguiente (llamada representación ordinal en [2] p. 214): Supóngase que se tienen 5 ciudades etiquetadas con las letras de la A a la E. Se define una cadena patrón que contenga todas las ciudades, por ejemplo: P = (A B C D E).

   Entonces un ciclo dado se denota usando P como referencia. Por ejemplo el ciclo:

   A – D – B – E – C

   Se denota como:

   (1 3 1 2 1)

   Lo que significa: Se toma la ciudad que aparece en el lugar número 1 de P (A) y cada vez que se toma una ciudad de P se elimina al mismo tiempo (con lo que el nuevo P es (B C D E)), luego se viaja de esa a la ciudad a la que aparezca en el lugar 3 en el P actual (la D; P es ahora (B C E)), ésta se conecta con la que aparece en el lugar 1 (B; P es (C E)), de ésta se viaja a la que está en el lugar 2 (E, en P queda únicamente C) y finalmente se viaja a la ciudad en el lugar 1 (C, la única).

En este caso, ¿es posible utilizar el operador de cruza de un punto de corte y generar cadenas válidas a partir de cadenas válidas? ¿Qué tanto se parecen, en general, los hijos a los padres bajo este esquema? ¿Qué tanto explota verdaderamente el AG el conocimiento adquirido?

5. Reformule el teorema del esquema para un AG sin elitismo que pretende resolver el problema del agente viajero para *n* ciudades, con los operadores especiales descritos en este capítulo y selección proporcional.

III.4 PROGRAMACIÓN

1. Modifique el AGS que programó en el ejercicio de programación 1 del capítulo 1 para que al menos conserve el mejor individuo de la generación, copiándolo a la siguiente. Pruebe su AGT con las funciones del ejercicio mencionado. Compare los resultados.

2. Pruebe el AGT con varias instancias del problema del agente viajero, castigando a las cadenas inválidas y utilizando los operadores convencionales (cruza de un punto y mutación uniforme).

3. Programe los operadores genéticos descritos en el texto y vuelva a probar el AGT con las mismas instancias del problema del agente viajero. Compare los resultados con los obtenidos en el ejercicio anterior.

III.5 REFERENCIAS

[1] Goldberg, D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989.
[2] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs,* 3a ed., Springer Verlag, 1996.

IV. ALGORITMOS GENÉTICOS NO CONVENCIONALES

IV.1 INTRODUCCIÓN

EN el capítulo 2 se analizaron los llamados algoritmos genéticos simples[5] (AGS). Ahí se demostró el teorema del esquema que nos permite determinar la manera en que un esquema dado se propaga en una población, bajo los operadores de selección proporcional, cruzamiento en 1 punto, y mutación uniforme. Obviamente existen otras posibilidades de manipular las poblaciones genéticas, como se discutió brevemente en el capítulo 3.

En este capítulo plantearemos un algoritmo idealizado (AGI), con el cual es posible subsanar ciertas deficiencias del AGS. Asimismo, mostramos que un AG constituye una cadena finita de Markov. De este hecho es posible establecer ciertos teoremas de convergencia. Tomando en consideración los requerimientos planteados del algoritmo genético idealizado y de las conclusiones derivadas de los teoremas mencionados, pasamos a describir un AG no convencional (es decir, con una dinámica diferente a la del AGS) que incluye: *a)* un esquema de selección determinística, *b)* elitismo, *c)* cruzamiento anular, *d)* ajuste paramétrico dinámico (auto adaptación), y *e)* un escalador de mutación aleatoria integrado. Con este algoritmo, que denominamos *ecléctico* (AGE), argumentamos que nos podemos aproximar a los requerimientos establecidos en al AGI y otros más no incluidos en éste.

IV.2 UN ALGORITMO GENÉTICO IDEALIZADO

Del análisis matemático planteado en el capítulo 2 y del análisis de los Caminos Regios, se pueden circunscribir las posibles fuentes de problemas de los AGs tradicionales a los siguientes elementos:

a) Los problemas engañosos.[6]
b) La correlación espuria.

Recordemos que la hipótesis de los bloques constructores[7] (HBC) sostiene que los AGs parten de bloques de corta longitud y poca especificidad para ir integrando bloques más específicos de longitud mayor.

---

[5] También conocidos como algoritmos genéticos canónicos [2].
[6] A los que también nos referiremos como problemas deceptivos.
[7] *Building Block Hypothesis.*

Un AG idealizado (AGI) que trabaja de acuerdo con la HBC ha sido definido en [1]. Este AGI trabaja con una cadena a la vez y, aunque no hay una población definida, está hecho para capturar las propiedades esenciales de un AG y satisfacer la HBC. El AGI trabaja como se describe a continuación:

ALGORITMO A.IV.2.1

1. En cada paso, elija una nueva cadena al azar, con probabilidad uniforme en cada bit.
2. La primera vez que se encuentre una cadena que contiene uno o más de los esquemas deseados, debe secuestrarse dicha cadena.
3. Cuando ocurra una cadena que contiene uno o más de los esquemas aún no descubiertos, instantáneamente debe cruzarse con la cadena secuestrada de forma tal que ésta contenga todos los esquemas deseados que se han descubierto hasta el momento.
4. Supongamos que los esquemas deseados consisten de N bloques de K 1's cada uno. ¿Cuál es el tiempo esperado (número de evaluaciones de función) hasta que la cadena secuestrada contenga todos los esquemas deseados? Sea p la probabilidad de encontrar el esquema deseado H en una cadena aleatoria ($p = 1/2^K$). Sea q la probabilidad de no encontrar H ($q = 1 - p$). El tiempo esperado para encontrar todos los N esquemas deseados está dado por:

$$E_N = \sum_{t=1}^{\infty} t((1-q) - (1-q^{t-1})^N)) \qquad \text{(IV.2.A)}$$

Esta ecuación se puede manipular algebraicamente para llegar a:

$$\begin{aligned} E_N &\approx -\frac{1}{p}\sum_{n=1}^{N}\binom{N}{n}\frac{(-1)^n}{n} \\ &= 2^K(\ln N + \gamma) \end{aligned} \qquad \text{(IV.2.B)}$$

El punto es que el AGI tiene un tiempo esperado que es del orden de $2^K \ln N$, en tanto que el Escalador de Mutación Aleatoria tiene un tiempo esperado del orden de $2^K N \ln N$; un factor de N más lento ($\gamma = 0.5772$, es la constante de Euler). Este análisis nos permite determinar cuándo un AG se desempeñará mejor que un escalador que, como se mencionó en el capítulo anterior, fue superior al AG tradicional en el caso de los Caminos Regios.

IV.2.1 Conclusiones

Del análisis de los Caminos Regios, el Escalador de Mutación Aleatoria y el AGI, concluimos que el AG se aproxima a un AGI si:

1. Las muestras son independientes. La población debe ser suficientemente grande, el proceso de selección debe ser suficientemente lento y la tasa de mutación lo suficientemente alta de manera que ningún *locus* se mantenga en un valor fijo en una mayoría significativa de las cadenas de la población.
2. Los esquemas deseados deben secuestrarse. La selección debe ser suficientemente fuerte para preservar los esquemas deseados que han sido descubiertos pero también suficientemente lenta para evitar correlaciones espurias en algunos esquemas altamente eficientes.
3. El cruzamiento debe ser "instantáneo". La tasa de cruzamiento debe ser tal que el tiempo de cruzamiento que combine dos esquemas deseados sea pequeño con respecto al tiempo que toma descubrir dichos esquemas.
4. La cadena debe ser grande. Suficientemente grande como para que el factor de aceleración *N*, en la ecuación (IV.2.B), sea significativo.

Los mecanismos delineados no son mutuamente compatibles en un AG tradicional. En la sección IV.4 se describe un AG que nos permite aproximarnos a las metas planteadas.

IV.3 MODELOS DE CADENAS DE MARKOV

En el capítulo 2 hicimos un análisis que depende del concepto de un *esquema*. Pudimos llegar a ciertas conclusiones interesantes respecto del AGS. Aquí enfocamos el comportamiento de los AGs modelándolos como cadenas de Markov finitas.

Las cadenas de Markov son procesos estocásticos en los cuales la probabilidad de que el proceso esté en el estado *j* en el tiempo *t* depende solamente del estado *i* en el tiempo *t-1*. Un "estado" de una población de un AG es, simplemente, una población particular. El conjunto de todos los estados es el conjunto de todas las posibles poblaciones de tamaño *n*. Éstas pueden ser numeradas en algún orden canónico indizado por *i*. Podemos representar la *i*-ésima de tales poblaciones con un vector $\vec{p}$ de longitud 2^l. El *y*-ésimo elemento de $\vec{p}$ es el número de apariciones de la cadena *y* en la población P_i. Bajo un AGS la población actual P_j depende sólo de la población en la generación previa. Por ello, el AG puede ser modelado como una cadena de Markov.

El conjunto de todas las poblaciones de tamaño *n* puede ser representado por una matriz Ξ en la que las columnas son todos los posibles vectores de población $\vec{p}_i$.

Hay

$$N = \binom{n + 2^l - 1}{2^l - 1} \qquad \text{(IV.3.A)}$$

poblaciones de tamaño *n*.

EJEMPLO E.IV.3.1

Sea $n = 2$ y $l = 2$. Las posibles poblaciones son

$$P_0 = \begin{pmatrix} 0\,0 \\ 0\,0 \end{pmatrix}, \quad P_1 = \begin{pmatrix} 0\,0 \\ 0\,1 \end{pmatrix}, \quad P_2 = \begin{pmatrix} 0\,0 \\ 1\,0 \end{pmatrix}, \quad P_3 = \begin{pmatrix} 0\,0 \\ 1\,1 \end{pmatrix}, \quad P_4 = \begin{pmatrix} 0\,1 \\ 0\,1 \end{pmatrix}$$
$$P_5 = \begin{pmatrix} 0\,1 \\ 1\,0 \end{pmatrix}, \quad P_6 = \begin{pmatrix} 0\,1 \\ 1\,1 \end{pmatrix}, \quad P_7 = \begin{pmatrix} 1\,0 \\ 1\,0 \end{pmatrix}, \quad P_8 = \begin{pmatrix} 1\,0 \\ 1\,1 \end{pmatrix}, \quad P_9 = \begin{pmatrix} 1\,1 \\ 1\,1 \end{pmatrix}$$

La matriz Ξ es

$$\Xi = \begin{matrix} 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 2 \end{matrix}$$

Un estado de una cadena de Markov corresponde a una columna de Ξ. Ahora podemos plantear una matriz de transición **P**. Ésta es una matriz de N **x** N, y cada elemento P_{ij} es la probabilidad de que la población P_j nacerá de la población P_i en un AGS.

Una cadena finita de Markov describe una trayectoria probabilística sobre un espacio finito *S* de cardinalidad $|S| = n$, en donde, como ya se mencionó, los estados pueden ser enumerados de 1 a *n*. La probabilidad $p_{ij}(t)$ de pasar del estado $i \in S$ al estado $j \in S$ en el paso *t*, se llama la *probabilidad de transición* de *i* a *j* en el paso *t*. Si las probabilidades de transición son independientes de *t*, es decir, $p_{ij}(t) = p_{ij}(s)$ para toda $i, j \in S$ y para todas las $s, t \in N$, se dice que la cadena de Markov es homogénea.

Las probabilidades de transición de una cadena de Markov finita y homogénea pueden ser plasmadas en la *matriz de transición* $\mathbf{P} = (p_{ij})$. Si para cada elemento, $P_{ij} \in [0,1]$ y $\sum_{j=1}^{|S|} p_{ij} = 1$ para toda $i \in \boldsymbol{S}$ la matriz se llama *estocástica*.

A continuación anotamos algunos teoremas que se pueden derivar del análisis de un AGS viéndolo como una cadena de Markov.[8]

TEOREMA IV.3.1

[8] Las demostraciones se pueden encontrar en [3].

El algoritmo genético simple no converge al óptimo global.

TEOREMA IV.3.2

En una cadena de Markov homogénea, el tiempo de transición entre un estado inicial *i* y cualquier otro estado *j* es finito, independientemente de los estados *i* y *j*.

En la sección IV.2.1, la conclusión 2 indica que, para aproximarnos a un AGI los esquemas deben ser secuestrados. Este es el principio que está detrás de los modelos elitistas. Esta conclusión intuitiva se formaliza en el siguiente teorema:

TEOREMA IV.3.3

El AGS que mantiene la mejor solución encontrada en el tiempo *después* de la selección, converge al óptimo global.

De manera análoga:

TEOREMA IV.3.4

El AGS que mantiene la mejor solución encontrada en el tiempo *antes* de la selección, converge al óptimo global.

En el caso del AGS la convergencia global no está garantizada. La razón es clara: para un AGS hay una probabilidad mínima acotada desde cero de perder el óptimo global en cada generación. Se sigue del lema de Borel-Cantelli [4], que este evento ocurre con probabilidad 1. Por otro lado, hay una probabilidad mínima de encontrar de nuevo una solución global si ésta se perdió, tal que este evento también ocurre con probabilidad 1. De hecho, la solución óptima global será encontrada y perdida con frecuencia infinita, de manera que la secuencia de mejores individuos encontrados en el tiempo por un AGS es una cadena de Markov irreducible en el espacio de estados $\{0, 1,..., n\}$, la cual no converge aunque la esperanza lo hace.

IV.3.1 Conclusiones

El análisis previo muestra que la convergencia global no es una propiedad inherente de un AGS. En otras palabras, el AGS original no puede ser considerado un algoritmo de optimización para problemas de optimización estática, porque es demostrable que no convergerá a ningún subconjunto de estados que contenga

cuando menos una solución global, aún en un tiempo infinito. La optimización estática, sin embargo, no era el propósito original al diseñar el AGS. De hecho, el interés se concentró en una estrategia que alcance una asignación óptima de intentos de manera que se minimicen las pérdidas esperadas en un medio ambiente incierto, posiblemente con recompensas variables en el tiempo. El *teorema fundamental de los AGs* no implica que el AGS convergerá al óptimo global en problemas de optimización estática. Además, debido a la irreducibilidad del AGS, es claro que no convergerá en absoluto.

No obstante, también se demostró que con el simple mecanismo de mantener el mejor individuo observado, podemos garantizar tal convergencia global. Este es un resultado muy importante que da una fundamentación sólida a la afirmación hecha en los párrafos iniciales, en el sentido de que los AGs son una alternativa robusta y general a los problemas de optimización.

Por tanto tenemos dos hechos demostrables:

a) Los AGs son una estrategia cuasióptima para atacar problemas *dinámicos*.
b) Los AGs elitistas garantizan convergencia global en problemas *estáticos*.

Sin embargo, como también ya se señaló, algunas técnicas de optimización tales como los escaladores, pueden ser más eficientes que los AGs en términos de qué tan eficientemente encuentran las soluciones estáticas ya mencionadas. Ahora quisiéramos discutir ciertos aspectos heurísticos que nos permiten incrementar la eficiencia de los AGs.

IV.4 MODELOS NO CONVENCIONALES

En este punto, debería ser claro que tenemos dos grandes campos de aplicación para los AGs: problemas de optimización *dinámica* y problemas de optimización *estática*. También ha sido mostrado que cualquiera de estos campos puede ser atacado exitosamente con la variante del AG apropiada. Nuestra preocupación, ahora, es encontrar una forma de hacer estas técnicas más eficientes. Modificando la estrategia básica de un AGS llegamos a lo que denominamos *modelos no convencionales de los algoritmos genéticos*.

IV.4.1 Elitismo

Como se discutió antes, cuando se mantiene una copia del mejor individuo podemos garantizar convergencia global para un problema de optimización dado. Hay variantes de modelos elitistas a las que hemos llamado *elitismo parcial* y *elitismo total*. Por elitismo parcial queremos decir que en una población de tamaño n mantenemos una copia de los mejores $\tau < n$ individuos hasta la genera-

ción *k*. Por elitismo total queremos decir que mantenemos una copia de los mejores *n* individuos hasta la generación *k*. Es decir, dado que hemos probado *nk* individuos hasta la generación *k*, nuestra población consistirá de los mejores *n* hasta ese punto.

En la figura F.IV.4.A mostramos el caso del elitismo total. Note que en la generación *k* hemos elegido los mejores *n* individuos posibles del total de los *nk* individuos considerados hasta ese punto. Al hacer esto, artificialmente forzamos a la población a orientarse hacia un conjunto focalizado de hiperplanos. En este contexto, un hiperplano es el espacio de búsqueda de las posibles soluciones al problema. El riesgo obvio es que restrinjamos el espacio de búsqueda de manera que sesgue el AG excesivamente. Para evitar este riesgo podemos alterar la estrategia de selección proporcional como se discute en lo que sigue.

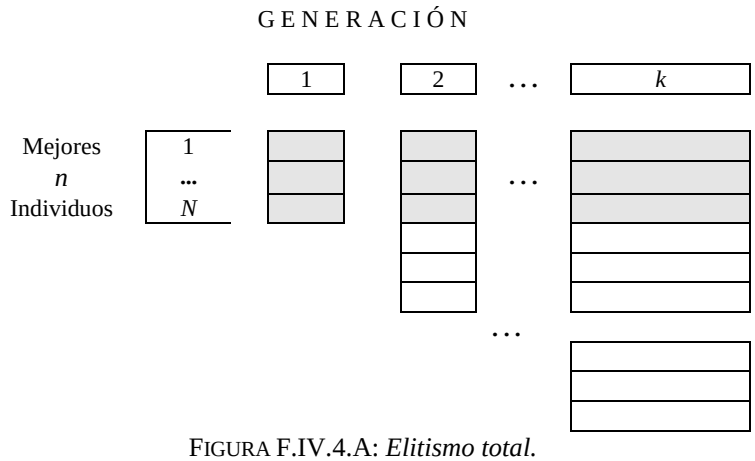


FIGURA F.IV.4.A: *Elitismo total.*

IV.4.2 Selección determinística

En el AGS hemos considerado selección proporcional. De hecho, el teorema del esquema ha sido derivado de tal consideración. Ahora consideramos la selección *determinística*. En este tipo de selección no consideramos el ajuste del individuo para determinar los descendientes más deseables. En su lugar, proponemos enfatizar la variedad genética imponiendo una estrategia que refuerce el cruzamiento de individuos predefinidos. Hay dos puntos de vista contrastantes. En uno de ellos, favorecemos que los genes del mejor individuo se crucen entre ellos mismos; en el otro favorecemos que los mejores individuos se crucen con los *peores*.

La primera estrategia se denomina el modelo de Nietzsche (MN), en donde los mejores elementos de la población se entremezclan en un esfuerzo por preservar los "mejores" genes. La otra estrategia se denominada el modelo de Vas-

concelos (MV), en donde los "mejores" individuos se mezclan con los "peores" individuos en un esfuerzo por explorar el más amplio espectro del espacio de soluciones.

IV.4.3 Modelo de Nietzsche

En este modelo adoptamos la estrategia de seleccionar determinísticamente el individuo *i* para cruzarlo con el individuo *i+1* para *i* = 1, 3, 5, ... Se supone que se ha aplicado elitismo total. Por ello, estamos forzando que los genes de los individuos 1 y 2 se crucen; que los genes de los individuos 3 y 4 se crucen, es decir, que los genes de los individuos *n-1* y *n* se crucen.

Un análisis superficial sugiere que un mecanismo en el cual se refuerzan las características benéficas de los individuos debe llevar a un mejor algoritmo. Sin embargo, la simulación del proceso no sustenta esta visión. La razón es, simplemente, que para los posibles esquemas de esta forma nos conduce a lo que se denomina *convergencia prematura*. Es decir, la selección determinística de los mejores individuos en esta forma no permite que haya una exploración lo suficientemente vasta del espacio de soluciones. En cierto sentido estamos restringiendo la fase de exploración en tanto maximizamos las características positivas de los individuos.

IV.4.4 Modelo de Vasconcelos

En este modelo adoptamos la estrategia de cruzar al individuo *i* con el individuo *n-i+1* determinísticamente. Como en MN, suponemos elitismo total. Así, adoptamos una estrategia que, superficialmente, destruye las características deseables de los mejores individuos cruzándolos deliberadamente con los peores. Sin embargo, cuando esto se hace en conjunción con elitismo total, esta estrategia conduce al análisis implícito de una más amplia variedad de esquemas (es decir, maximiza la exploración del espacio de soluciones). La exploración de tal espacio se enfoca vía el elitismo total implícito en el modelo. En el pasado [5] se han planteado modelos que persiguen metas similares. No obstante, en el caso de Eshelman la variedad se busca haciendo un tipo de cruzamiento llamado de *recombinación destructiva*.

El MV permite al algoritmo atrapar los mejores esquemas sin restringir el espacio de búsqueda en una forma sensible. Así podemos aproximar al AGI ya mencionado. Ahora hacemos un breve repaso de los requerimientos para que un AG se aproxime a un AGI:

a) Ningún *locus* se debe fijar a un valor en un número grande de las cadenas de la población.

Esto se logra porque determinísticamente estamos afectando a los posibles *locus* problemáticos con una cruza mejor-peor.

b) La selección debe ser suficientemente fuerte para que preserve a los esquemas deseables pero también debe evitar correlación espuria en los esquemas altamente efectivos.

Debido a que trabajamos con elitismo, preservamos los esquemas deseados y, como antes, evitamos correlación espuria al cruzar individuos disímiles.

c) La tasa de cruzamiento debe ser tal que el tiempo de cruza que combine a dos esquemas sea pequeño con respecto al tiempo que toma descubrir tales esquemas.

El elitismo total garantiza que, aún en el caso del MV, el peor individuo de la población *k* está en el estrato que corresponde al 1/*k* porcentual. Por ejemplo, si $n = 50$ y observamos la 20^{ava} población, los individuos en ésta están dentro del mejor 5% del total de los individuos analizados. Esta característica es incrementalmente expuesta a medida que el AG avanza.

d) La cadena debe ser suficientemente grande para que el factor de aceleración *N* sea significativo.

Este es el único elemento que no podemos garantizar. Es decir, para *N*s relativamente pequeñas un escalador puede desempeñarse mejor que un AG. Con respecto a esto tendremos más que decir cuando hablemos del algoritmo genético ecléctico.

IV.4.5 Autoadaptación

Al correr un AG hay varios parámetros que deben determinarse *a priori*: Tres de ellos son los más comunes: *a)* la tasa de cruzamiento (P_c), *b)* la tasa de mutación (P_m), y *c)* el tamaño de la población (N).

En muchos casos el usuario trata de sintonizar estos parámetros haciendo una serie de corridas en diferentes problemas "representativos" (por ejemplo, De Jong [6], Galaviz [7]). Ahora discutimos con más detalle la *autoadaptación* de los parámetros mencionados. En un AG autoadaptable los tres parámetros se incluyen como una extensión del genoma de manera tal que éstos evolucionan junto con el individuo. Esta es una práctica común en las estrategias evolutivas [8].

La idea detrás de la autoadaptación es que el AG no solamente explore el espacio de soluciones sino también el espacio de parámetros. De esta manera, el genoma se divide en dos subgenomas: un subgenoma de estrategias y un subgenoma de solución. En términos de un AG clásico, el subgenoma de solución corresponde a lo que hasta este punto ha sido denominado simplemente el genoma. Ambos subgenomas son sujetos de los operadores genéticos. Deberíamos considerar al subgenoma de parámetros como un conjunto de tres subgenomas que son funcionalmente independientes. El genoma autoadaptable se muestra en la figura F.IV.4.B. Note que el tamaño de la población (*N*) se trata implícitamen-

te considerando no el tamaño de la población misma, sino el número de descendientes producto del cruzamiento.

Recordemos, por ejemplo, que en los Caminos Regios el número de descendientes se manejó vía truncamiento sigmoidal. Ahí tenemos una estrategia de autoadaptación en embrión, ya que el parámetro que determina los detalles del número de descendientes es elegido *a priori*.

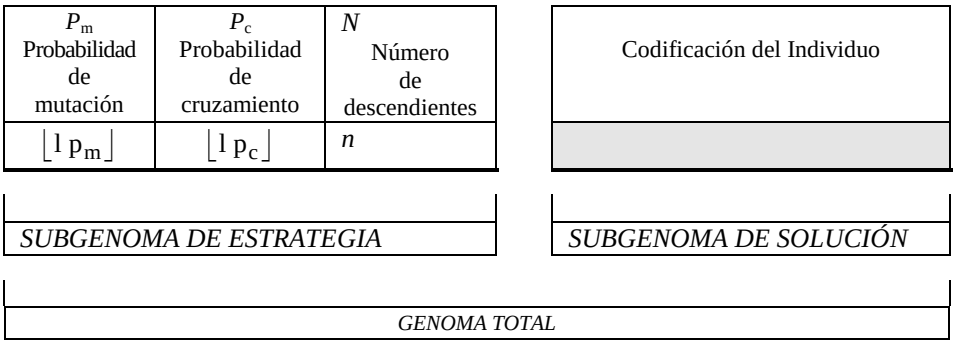


FIGURA F.IV.4.B: *Genoma autoadaptable.*

IV.4.5.1 Algoritmo genético autoadaptable individual

En este caso, los diferentes parámetros toman sus valores de consideraciones que afectan a los individuos considerados como unidades independientes, en contraste con un enfoque de los individuos como miembros de la población (véase la siguiente sección). Por ejemplo, podemos codificar los *parámetros de control* como sigue:

a) Probabilidad de mutación

> Sea $p_m \in (0,1]$ la probabilidad de mutación de un bit para cierto individuo. p_m se representa en binario como un entero n en donde $n = \lfloor K\, p_m \rfloor$ y $K \in N$. Hay N tasas de mutación que denotamos por $(p_m)_i\ i=1,2,...,N$. Estas evolucionan en el tiempo bajo los operadores del AG. Cada individuo se hace mutar, entonces, con probabilidad $(p_m)_i$.

b) Probabilidad de cruzamiento

> Sea $p_c \in [0,1]$ la probabilidad de cruzamiento para un cierto individuo. Como en el caso de p_m, p_c se codifica en binario como un entero n en donde $n = \lfloor K\, p_c \rfloor$. El código de p_c es también parte de la cadena genética y

sujeto a los operadores genéticos. Dados dos individuos con probabilidades de cruzamiento p_{c1} y p_{c2}, éstos se cruzan con probabilidad:

$$p_{c12} = \frac{p_{c1} + p_{c2}}{2}$$

c) Número de descendientes

Cada individuo tiene un número de descendientes que también se codifica en una cadena genética y es sujeto de los operadores genéticos. Sean c_1 y c_2 los descendientes de dos individuos. Cuando tales individuos se cruzan, el número de descendientes que generan está dado por:

$$n = \left(\frac{c_1 + c_2}{2} \right) \left(\frac{\overline{f_{1,2}}}{f_{\max}} \right)$$

en donde $\overline{f_{1,2}}$ es el ajuste medio de estos dos individuos y $f_{\max}$ es el mejor ajuste para la población en la generación presente. El número de descendientes determinado como se ha descrito, propende a que las estructuras genéticas pobres tengan menor probabilidad de sobrevivir que las mejores.

IV.4.5.2 Algoritmo genético autoadaptable poblacional

En este caso de autoadaptación, la forma en que los operadores afectan el desempeño del AG considera a la población (para cualquier generación dada) como un todo. En un sentido, este AG autoadaptable apunta a mejorar los valores promedio de la población y no a los valores de cada individuo.

a) Probabilidad de mutación

Como en el caso de la autoadaptabilidad individual, p_m se codifica en cada individuo. Aquí, sin embargo, la tasa de mutación para toda la población en la k-ésima generación g_k se calcula como sigue:

$$(p_m)_k = \frac{1}{N} \sum_{i=1}^{N} (p_m)_i$$

Por tanto, la tasa de mutación está fija para todos los individuos de la población durante g_k.

b) Probabilidad de cruzamiento

p_c está, como antes, codificado en el genoma de cada individuo. Ahora, en la generación g_k la tasa de cruzamiento está dada por

$$(p_c)_k = \frac{1}{N}\sum_{i=1}^{N}(p_c)_i$$

Aquí, de nuevo, la tasa de cruzamiento está fija para todos los individuos durante g_k.

c) Número de descendientes

Como anteriormente, n, el número de descendientes, se codifica en el genoma de cada individuo y, como antes, el número de descendientes n_k en la k-ésima generación está dado por

$$n_k = \frac{1}{N}\sum_{i=1}^{N} n_i$$

Como en los dos casos precedentes, el número de descendientes es fijo para todos los individuos durante g_k.

Al usar una estrategia autoadaptiva se evita una elección de parámetros arbitraria, hasta cierto punto. Es usual establecer cotas superiores en los posibles valores codificados en p_m, p_c y n. En ese sentido, aún hay una elección arbitraria de los valores iniciales de los parámetros. Sin embargo, los individuos que representan los mejores parámetros para un problema en particular se pueden *aprender* del problema que está siendo resuelto. Se ha demostrado que la alternativa autoadaptiva compite favorablemente (véase [7], p. 162) con AGs cuyos parámetros se establecen de manera tradicional. Tal vez el punto más importante de las estrategias autoadaptivas es el siguiente. Durante el transcurso de un AG los parámetros de los operadores genéticos tienen valores óptimos que «no se mantienen constantes» a lo largo del proceso. Es ahí donde un AG autoadaptivo se muestra superior, pues va adecuando su comportamiento de manera dinámica.

También hay que hacer notar que aunque típicamente la población se mantiene fija de generación en generación, el número variable de descendientes permite que el número de individuos evaluados de generación en generación se ajuste dinámicamente a las condiciones del problema.

IV.4.6 Un algoritmo genético ecléctico

En este momento tenemos todos los elementos, tanto teóricos como prácticos para proponer lo que llamaremos un *algoritmo genético ecléctico* (AGE). El eclecticismo aquí se refiere al hecho de que estamos dispuestos a adoptar las estrategias que consideramos mejores, independientemente del problema. De hecho, llegamos a un algoritmo híbrido: estrictamente no es un simple AG. El AG que será discutido es aplicable a una amplia gama de problemas sin la necesidad de hacer consideraciones particulares. De hecho, usaremos este AGE para resolver problemas tanto numéricos como no numéricos. Los aplicaremos a la optimización y al aprendizaje. Esto se logrará sin cambiar el AGE y diseñando, simplemente, las funciones de ajuste. A lo largo del texto hemos enfatizado el hecho de que los AGs son flexibles, fáciles de ajustar al problema en turno, robustos y capaces de llevar a cabo búsquedas globales en el espacio de soluciones. También mostramos cómo se aproximan a una herramienta casi óptima para problemas dinámicos y que su convergencia está garantizada en problemas estáticos (vía elitismo).

La desventaja que se cita frecuentemente es que los AGs, como todos los métodos "débiles" de inteligencia artificial, incorporan el conocimiento del problema en la estructura del método mismo, en vez de representar las estrategias implicadas explícitamente. En un AG el "conocimiento" dependiente del problema se representa por los valores que deben ubicarse en sus operadores y en los valores que éstos toman.

Hemos dado argumentos teóricos que explican las desventajas de los AGs y, por tanto, nos permiten tomar las previsiones necesarias para subsanarlas. Vía el AGI pudimos determinar las características deseables que un AG eficiente debe encarar para evitar la decepción y la correlación espuria.

Aún cuando podemos determinar las causas para un desempeño del AG debajo del óptimo, todavía nos encontramos con el problema de ajuste de parámetros. La elección de los parámetros de control de los AGs sigue siendo un asunto abierto a la fecha. Varios investigadores han propuesto parámetros de control que garantizan un buen desempeño en un conjunto base de prueba de funciones objetivo elegido cuidadosamente. Dos conjuntos de parámetros han emergido en el uso común de los AGs. Uno de ellos tiene una población de tamaño reducido y probabilidades de mutación y cruzamiento relativamente altas. El otro tiene una población de tamaño mayor, pero probabilidades de mutación y cruzamiento mucho menores. Típicos de estas dos categorías son: *a)* $p_c = 0.9$, $p_m = 0.01$, $N = 30$[9] y *b)* $p_c = 0.6$, $p_m = 0.001$, $N = 100$.[10] Mientras que el primer conjunto claramente enfatiza el rol de la mutación, el segundo la minimiza. La alta tasa de cruzamiento en el primer conjunto también indica que un alto porcentaje de perturbación de los esquemas es deseable en poblaciones reducidas.

[9] Sugerido por Grefenstette.
[10] Sugerido por De Jong.

Una alternativa interesante es dejar que el algoritmo mismo decida acerca de los parámetros anteriores, a través de la autoadaptación. En lo que sigue hacemos una breve síntesis de estos asuntos y agregamos un elemento hibridizante que nos permite subsanar el problema del factor *N* que se mencionó en IV.2.1.

IV.4.6.1 Selección, elitismo y autoadaptación

El AGE incluye las siguientes componentes:

a) Selección determinística
b) Elitismo
c) Autoadaptación

Ya hemos discutido las posibilidades y ventajas de cada uno de estos elementos en las secciones precedentes. Sólo resta definir la variación específica de cada componente.

La selección se hace de acuerdo con la estrategia de Vasconcelos: cruzando el individuo *i* con el individuo *N-i+1* para $i = 1, 2, ..., N$.

El elitismo es total, reteniendo siempre los mejores *N* individuos de los *Nk* examinados hasta la generación *k*.

La autoadaptación es sobre la población. Tres parámetros se codifican y autoadaptan: mutación, cruzamiento, y número de descendientes.

La mutación es uniforme.
El cruzamiento es anular.

En un esquema tradicional de AGs, los contenidos genéticos de los esquemas se intercambian seleccionando aleatoriamente el punto de cruza X_p y luego intercambiando los $(l-X_p)$ bits de la extrema derecha. Por ejemplo, si $H_1(t) = 100101$, $H_2(t) = 101010$ y $X_p = 3$, después de un cruzamiento de 1 punto las cadenas resultantes son $H_1(t+1) = 100010$ y $H_2(t+1) = 101101$. La supervivencia del esquema está determinada por esta forma de operar y corresponde a la expresión $1 - p_c \frac{\delta(H)}{l-1}$.

El cruzamiento desarrollado en este caso es el siguiente: *a)* se estipula el tamaño de un segmento de cruzamiento (Ξ); *b)* la posición de cruzamiento se elige aleatoriamente (X_p); *c)* el material genético se intercambia tratando ambas cadenas como anillos. Por ejemplo, si $\Xi = 3$; $H_1(t) = 100101$, $H_2(t) = 101010$ y $X_p = 4$. Después del cruzamiento en anillo, las cadenas resultantes son $H_1(t+1) = 100110$ y $H_2(t+1) = 101001$.

El cruzamiento en anillo se ilustra en la figura F.IV.4.C:

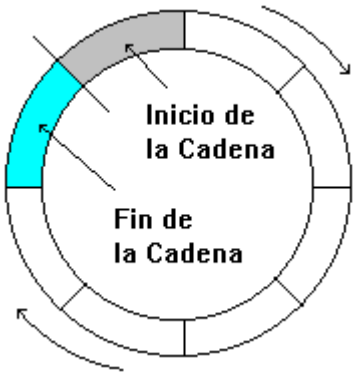


FIGURA F.IV.4.C: *Genoma anular.*

Aquí, la probabilidad de supervivencia del esquema dado se puede encontrar como sigue:

a) Un esquema sobrevive al cruzamiento anular si

$$\delta(H) < l_r$$

en donde $l_r \equiv$ longitud del anillo

y entonces sobrevive con $P_s|_{\delta(H)<l_r} = \dfrac{l_r - \delta(H)}{l}$

b) Hay 3^l posibles esquemas de longitud l.

c) Los esquemas de longitud N_s menores que l_r están dados por

$$N_s = \sum_{i=0}^{l_r} (l - i) 3^{i+1}$$

$$N_s = \sum_{i=0}^{l_r} l \bullet 3^{i+1} - \sum_{i=1}^{l_r} i \bullet 3^{i+1}$$

Entonces podemos escribir

$$P_s = \frac{(l_r - \delta(H)) N_s}{3^{l + \log_3 l}}$$

En el cruzamiento anular el genoma ya no se ve como una colección de bits sino como un anillo cuyo bit de la extrema izquierda es contiguo al bit de la extrema derecha. Cuando se aplica cruzamiento anular hay dos parámetros que considerar para cada intercambio:

a) El *locus* de inicio del intercambio.

b) La longitud del semi-anillo.

Un ejemplo de este tipo de cruzamiento se ilustra en la figura F.IV.4.D.

Como ya se mencionó, este algoritmo se acerca al comportamiento del AGI. Al incluir un mecanismo autoadaptivo, modifica su comportamiento sin impactar negativamente en sus características deseables.

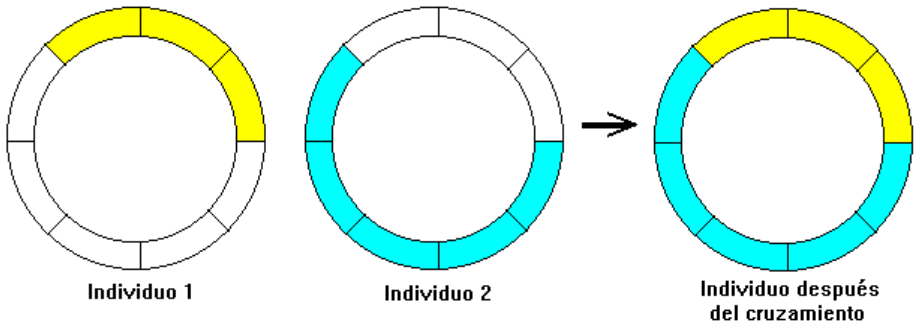


FIGURA F.IV.4.D: *Cruzamiento anular*.

IV.4.6.2 Escalador adaptivo

Como se mencionó en el capítulo 2, el Escalador de Mutación Aleatoria (RMHC, por sus siglas en inglés) es capaz de superar al AGS para ciertas funciones. Para aprovechar las características de un escalador en tales casos, incluimos un RMHC como parte del algoritmo. Es decir, el AGE consiste de un AG autoadaptable más un RMHC. ¿Cómo determinamos cuándo el RMHC debe activarse en lugar del AG propiamente dicho? Hacemos esto con un mecanismo autoadaptivo que describimos a continuación.

a) El primer paso es describir dos cotas:
1) El porcentaje mínimo de escalamiento (η_λ).
2) El porcentaje máximo de escalamiento (η_μ).

en donde

$$0 < \eta_\lambda < \eta_\mu \leq 1$$

El algoritmo RMHC se activará, al menos, η_λ del tiempo y cuando mucho η_μ del tiempo.

b) Como segundo paso, debemos definir otras dos cotas:
1) El mínimo número de evaluaciones de la función objetivo que deseamos efectúe el RMHC en cada invocación (η_λ).
2) El número máximo de tales evaluaciones (η_μ).

Estas dos cotas se dan como un porcentaje del tamaño de la población *N*. El porcentaje efectivo de las evaluaciones de función del RMHC (relativo a *N*) se incluye en el subgenoma de estrategias y es sujeto de los operadores genéticos.

El número efectivo de evaluaciones por invocación del RMHC N_η en la generación *k* está dado por

$$(N_\eta)_k = \frac{1}{N}\sum_{i=1}^{N}(N_\eta)_i$$

c) En la generación g_k la eficiencia del escalador es evaluada de

$$\eta_\phi = \frac{1}{N}\sum_{i=1}^{N}(\iota_\eta)_i$$

donde

$$\iota_\eta = \begin{cases} 1 \;\; si\, el\, individuo\, se\, encontró \;\; por\, el\, RMHC \\ 0 \;\; en\, otro\, caso \end{cases}$$

Para poder determinar el valor de ι_η debemos incluir un nuevo elemento en el genoma. Este elemento es de tipo *booleano*. Será activado (*cierto*) cuando el individuo haya nacido del escalador y desactivado en otro caso.

El genoma del AGE se muestra en la figura F.IV.4.E. En él podemos encontrar todos los elementos del esquema autoadaptivo *y* del algoritmo RMHC.

P_m Probabilidad de mutación	P_c Probabilidad de cruzamiento	N Número de descendientes	ι_η Originado por RMHC	Evaluaciones de RMHC	Codificación del individuo
$\lfloor l\, p_m \rfloor$	$\lfloor l\, p_c \rfloor$	n	η	N_η	
ESTRATEGIA					SOLUCIÓN

FIGURA F.IV.4.E: *Genoma autoadaptivo para el algoritmo genético ecléctico.*

d) Denotando la probabilidad de invocar el RMHC con η_τ, tenemos:

$$\eta_\tau = \begin{cases} \eta_\lambda & si \;\; \eta_\phi < \eta_\lambda \\ \eta_\phi & si \;\; \eta_\lambda \le \eta_\phi \le \eta_\mu \\ \eta_\mu & si \;\; \eta_\phi > \eta_\mu \end{cases}$$

e) Generar un número aleatorio ρ_K, donde $0 < \rho_K \le 1$. Invocar el RMHC si $\eta_\tau \le \rho_K$.

f) Una vez que el RMHC se programa para iniciar, la cadena en la cual operará es elegida aleatoriamente de los primeros cinco individuos de la población. Recordemos que los individuos en la población están ordenados del mejor (individuo 1) al peor (individuo *N*). Por tanto, para elegir la cadena ι_η en la cual RMHC operará, hacemos

$$\iota_\eta = \lfloor Rx5 \rfloor + 1$$

Donde *R* es un número aleatorio uniformemente distribuido y $0 < R \leq 1$.

El resultado de la estrategia descrita es garantizar que el RMHC se activará cuando haya probado su efectividad. Sin embargo, la probabilidad de que el RMHC reemplace al AG es acotada desde arriba por η_μ. Esto evita que el RMHC se adueñe del proceso. Por otra parte, el RMHC será invocado con probabilidad η_λ lo cual, a su vez, evita la posibilidad de que debido a la baja eficiencia del RMHC en alguna generación g_k, éste se inhabilite por el resto del proceso. En esencia, por tanto, el AGE incorpora un proceso de RMHC que es autoadaptivo en dos sentidos: *a)* porque su actividad está determinada por su eficiencia, y *b)* porque el número adecuado de evaluaciones de la función evoluciona a medida que el algoritmo se desarrolla.

Se señaló anteriormente que el nombre "escalador" se refiere al hecho de que estos procesos, se supone, atinan en puntos de optimalidad cuando el algoritmo ha alcanzado un espacio de vecindad en el espacio de solución. Aquí, sin embargo, el RMHC tiene dos funciones: *a)* realmente atina en máximos locales cercanos, y *b)* refuerza la variedad de la población al explorar nuevos esquemas que el AG, de otra forma, pasaría por alto. El hecho más significativo acerca de este algoritmo de modo mixto (GA- RMHC) es que el AG es el que hace la búsqueda fina de los óptimos locales, y el RMHC funciona como un agente disparador que localiza soluciones subóptimas muy eficientemente.

Note que en un AG como el descrito, claramente el teorema del esquema no es aplicable. De hecho, pocos de los criterios empleados allí que tienen relevancia en la forma en que el AGE puede ser analizado. Como ya se dijo, el tratamiento teórico de los AGs está aún en fase de desarrollo y la generalización de los conceptos derivados de un análisis de esquemas o de cadenas de Markov aún no existe. Esto no significa que no podamos aprovechar la teoría ya desarrollada. Ésta nos da una manera eficiente de establecer comparaciones entre las diferentes variaciones de AGs. Por ello será útil establecer una medida de desempeño relativo para evaluar el algoritmo ecléctico y otras variaciones.

Aunque el teorema del esquema se encuentra lejos del espíritu del AGE, podemos mencionar, si bien cualitativamente, cómo se compara con el AGI, cuyas características deseables se identificaron de un análisis de esquemas tradicional. La incorporación de autoadaptación y escalamiento del AGE se aproxima al comportamiento del AGI ya discutido, ya que:

a) Vía MV no hay valores fijos en un *locus* específico.
b) Vía elitismo total, los esquemas deseables son secuestrados para preservar las mejores características.
c) Vía autoadaptación, el cruzamiento se escoge para mantener dinámicamente la mejor tasa de cruzamiento.
d) El factor de aceleración es, al menos, igual al del escalador debido a que el escalador se incluye explícitamente.

IV.5 EJERCICIOS

1. La convergencia de un algoritmo genético elitista (ver teoremas IV.3.3 y IV.3.4) puede interpretarse, intuitivamente, si se hace un bosquejo del proceso genético. Haga tal bosquejo: *a)* suponga un proceso de 500 generaciones; *b)* grafique el número de generación en el eje horizontal y el desempeño del algoritmo (función de *fitness*) en el eje vertical; *c)* trace una línea horizontal que represente *S*, la solución al problema (valor máximo de desempeño). Note que, en un AGS el desempeño se comporta como una línea que parte de (0,a) [a << 1] y se aproxima a *S* pero no de forma monotónica. ¿Cómo se comporta un AG elitista? ¿Qué diferencia existe cuando se incluye el mejor individuo *antes* de la selección vs. el caso en el que se incluye el mejor individuo *después* de la selección?

2. En las denominadas estrategias de Vasconcelos y Nietzsche se aplican dos criterios: *a)* apareo determinístico; *b)* elitismo total. Existen cuatro combinaciones que se muestran en la tabla T.E.2.A.

TABLA T.E.2.A: *Estrategias determinísticas.*

	Apareo	Elitismo
A	I *vs.* i+1	Ninguno
B	I *vs.* i+1	*Total*
C	i *vs.* N-i+1	Ninguno
D	i *vs.* N-i+1	*Total*

Discuta el comportamiento de los cuatro posibles AGs definidos en la tabla. ¿Qué esperaría, en función de los resultados discutidos en la sección 4? Identifique, en la tabla, las estrategias de Vasconcelos y de Nietzsche.

3. Una estrategia alterna para la *autoadaptación* es que, periódicamente, un meta algoritmo (externo al AG propiamente dicho) se active y se encargue de actualizar los parámetros del proceso (P_c, P_m y N). Discuta las distintas alternativas para colectar las estadísticas del AG de manera que el meta algoritmo pueda tomar una buena decisión en cuanto a los valores de los parámetros del sistema.

4. En el escalador adaptivo hay dos parámetros que determinan el porcentaje mínimo y máximo de actividad del mismo. Discuta si los valores de 1/8 y 1/2 son apropiados. ¿Qué efecto tendría reemplazar los valores anteriores por 1/8 y 3/4? ¿Por 1/16 y 1/2?

5. Un AG es una cadena de Markov a la cual le corresponde una matriz de transición estocástica. Indique cuál es el significado de este hecho, desde el punto de vista de la convergencia del algoritmo. (Sugerencia: Piense en lo que pasa cuando la población inicial de un AG es aleatoria contra lo que pasa cuando dicha población es generada de acuerdo con las características del problema). (Sugerencia: vea el ejercicio de programación 3.)

6. Invente una estrategia de autoadaptación que utilice metafunciones objetivo de la siguiente manera: *a)* los valores de P_m deben cambiar para cada generación, en función de como P_m afecta la velocidad de convergencia del algoritmo; *b)* los valores de P_c deben cambiar para cada generación, en función de como P_c afecta la velocidad de convergencia del algoritmo; *c)* los valores de N deben cambiar para cada generación, en función de como N afecta la velocidad de convergencia del algoritmo. ¿Qué tipo de estadísticas debe llevar para determinar lo anterior? Para cada uno de los tres parámetros se requiere una función de adaptación distinta. Sugiera al menos una.

IV.6 PROGRAMACIÓN

1. Escriba un programa que implemente un escalador de mutación aleatoria. Prográmelo para que resuelva la función de Camino Regio que se discutió en el capítulo 2. Compare sus resultados con los resultados teóricos esperados de (IV.2.B).
2. Escriba un programa que: *a)* implemente la estrategia de Nietszche con elitismo de un solo individuo; *b)* implemente la estrategia de Vasconcelos sin elitismo. ¿Qué resultados observa? ¿Cuál es mejor?
3. En una matriz estocástica (aquella en la que la suma de los valores de cada una de las filas es igual a 1) se cumple que la multiplicación iterada de dicha matriz converge a un conjunto de vectores iguales. Cada uno de estos vectores se denomina el "punto fijo" de la matriz. Escriba un programa que itere la siguiente matriz Ξ y demuestre que converge a los valores del punto fijo ς.

$$\Xi = \begin{pmatrix} .1 & .1 & .1 & .1 & .6 \\ .2 & .2 & .2 & .2 & .2 \\ .12 & .34 & .45 & .001 & .089 \\ .1 & .2 & .3 & .33 & .07 \\ .1 & .2 & .3 & .2 & .2 \end{pmatrix}$$

$$\varsigma = (\ 0.1288763500, 0.2286794500,\ 0.2958232750, 0.1472650032,\ 0.1993559218\ )$$

Representación de números en un genoma

Al trabajar problemas de optimización numérica es necesario codificar conjuntos de números en el genoma de un AG. Una posibilidad es representar cada número como una secuencia binaria con el siguiente formato. El bit más a la izquierda corresponde al signo (*1* = negativo); *E* bits determinan la parte entera del número; *D* bits corresponden a la parte decimal de tal número. *E* y *D* son elegidos de manera que cubran el rango de valores del dominio de la función con que se está trabajando. Suponga que *E* = 2 y *D* = 25. Claramente, el tamaño del genoma (γ) está dado por $\gamma = \zeta\ x(\ E + D + 1\ )$, en donde ζ denota el número de variables.

Tomando "/" como un separador, la cadena

0/00/01000000000000010101101100

corresponde al número 0.4999520481. El genoma se forma, simplemente, concatenando las variables. Es fácil ver cómo se puede regular el rango de valores. X_i está dado por

$$-2^{E+1} + 2^{-D} \le X_i \le +2^{E+1} - 2^{-D}$$

o, aproximadamente,

$$-2^{E+1} \le X_i \le +2^{E+1}$$

Esta representación nos permite trabajar con números reales o enteros (haciendo *D* = 0) y regular la precisión incrementando/decrementando el tamaño de *D* cuando se trabaja con números reales.

4. Escriba un programa que resuelva el siguiente problema:
   Minimizar

$$f(\ \bar{x}\ ) = 100(\ x_2 - x_1^2\ )^2 + (\ 1 - x_1\ )^2$$

   sujeto a las siguientes restricciones

$$c_1 : x_1 + x_2^2 \geq 0$$
$$c_2 : x_1^2 + x_2 \geq 0$$
$$-0.5 \leq x_1 \leq 0.5$$
$$x_2 \leq 1.0$$

Considere que el genoma es una cadena como se describió arriba.

Invente una forma de asignar valores a la función objetivo cuando los valores del genoma no satisfagan las restricciones.

b) Use un AGS
c) Use un AG elitista

Reporte sus resultados y contrástelos.

5. Repita el problema anterior, pero ahora:

Use un AG elitista.
Use un AG que siga la estrategia de Vasconcelos.
Reporte sus resultados y contrástelos.

IV.7 REFERENCIAS

[1] Mitchell, M., "An Idealized Genetic Algorithm", *Introduction to Genetic Algorithms*, MIT Press, 1996, pp. 132-138.
[2] Goldberg, D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989, pp. 40-41.
[3] Rudolph, G., "Convergence Analysis of Canonical Genetic Algorithms", *IEEE Transactions on Neural Networks*, 5(1):96-101, enero, 1994.
[4] Feller, W., *An Introduction to Probability Theory and Its Applications*, John-Wiley & Sons, 1970.
[5] Eshelman L., "The CHC Adaptive Search Algorithm: How to have Safe Search when Engaging in Nontraditional Genetic Recombination", en *Foundations of Genetic Algorithms*, Rawlins, G., Ed., Morgan Kauffman, 1991.
[6] De Jong, K., *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, doctoral dissertation, University of Michigan, 1975.
[7] Galaviz, J., y Kuri, A., "A Self-Adaptive Genetic Algorithm for Function Optimization", *Proceedings ISAI/IFIPS*, 1996, p. 156.
[8] Bäck, T., Hammel, U., y Trojanowski, K., "Evolutionary Computation: Comments on the History and Current State", *IEEE Trans. on Evolutionary Computation*, Vol. 1, No. 1, 1997, p. 3.

V. ALGORITMOS GENÉTICOS APLICADOS AL APRENDIZAJE

V.1 INTELIGENCIA Y APRENDIZAJE

COMO ya se ha demostrado, los AGs son herramientas versátiles, flexibles y sólidas cuando se aplican a la optimización. Aunque, en el fondo, es posible entender los problemas de aprendizaje como una variante más de un problema de optimización, un enfoque rígido a este problema establece restricciones que lo hacen de difícil solución. Puesto que una de las metas fundamentales de la inteligencia artificial es la del aprendizaje automatizado, la aplicación de los AGs al aprendizaje se identifica como una meta básica. Las capacidades predictivas se reconocen como una característica fundamental del comportamiento inteligente (definido en función de un comportamiento adaptivo, dado que un organismo inteligente debe anticipar los eventos para adaptar su comportamiento a la luz de una meta).

En este capítulo discutimos cuatro enfoques distintos al problema del aprendizaje automático usando algoritmos genéticos:

a) Los algoritmos genéticos coevolutivos.
b) Los autómatas genéticos.
c) Los sistemas clasificadores evolutivos.
d) El ajuste paramétrico de orden libre.

Los tres primeros enfoques corresponden a lo que, en inteligencia artificial clásica, se denomina aprendizaje supervisado. El último corresponde a técnicas de aprendizaje no supervisado. En este último caso, se aplica una técnica genética al espacio de configuración de las soluciones en vez de a las soluciones mismas. A este tipo de enfoque se le ha denominado, en ocasiones, *algoritmos genéticos de orden*.

V.2 ALGORITMOS GENÉTICOS COEVOLUTIVOS

El concepto de coevolución, estableciendo un símil con la naturaleza, se basa en la observación de que los individuos dentro de dos especies tienden a reforzar mutuamente las características de aptitud una en función de la otra. De hecho, las especies no han evolucionado de manera aislada y es la competencia entre ellas la que ha llevado a la proliferación de la variedad biológica. Esto último se

puede interpretar como una búsqueda de nuevas soluciones a los problemas del entorno y las correspondientes mejoras sustantivas. La competencia entre dos especies obliga a los individuos de ambas especies a evolucionar de manera positiva.

Desde el punto de vista individual, la competencia da origen a procesos simbióticos cuando ambas especies se benefician y a procesos parasíticos cuando solamente una de ellas lo hace.

El proceso de aprendizaje puede ser definido como la optimización de una estrategia de solución a un problema dado. Por ejemplo, cuál es la mejor estrategia para: ¿invertir en la bolsa de valores?, ¿establecer reglas de intercambio económico?, ¿diseñar un ala?

En los ejemplos anteriores hay dos características básicas del problema: *a)* que existe un número suficientemente grande de alternativas factibles de manera que, para todo efecto práctico, es ilimitado, y *b)* que debemos seleccionar, de estas posibles alternativas, una que sea la mejor entre todas, en términos relativos. La característica (a) nos obliga a determinar el subconjunto contra el cual debemos establecer nuestra medida de eficiencia *a priori*. La característica (b) nos obliga a establecer una medida de eficiencia de la estrategia relativa a un subconjunto (necesariamente finito y acotado) del total de estrategias a nuestro alcance. Es decir, de un conjunto N ($N \to \infty$) debemos escoger n ($n << N$) estrategias contra las cuales comparar el desempeño de las estrategias candidatas. Pero esto, aparentemente, induce a un círculo vicioso. Si definimos el conjunto de estrategias *a priori*, ello presupone un criterio de eficiencia que antecede al criterio de optimización.

Para evitar este círculo vicioso planteamos el siguiente algoritmo:

V.2.1 Algoritmo de coevolución

1. Elegir el tamaño de una muestra (M) de estrategias.
2. Del espacio de posibles estrategias escoger dos de éstas al azar.
3. Hacer que las dos estrategias elegidas compitan entre sí.
4. Cada una de ellas recibe un criterio de desempeño.
5. Repetir los pasos 2, 3 y 4 un número predeterminado de veces.
6 Seleccionar las N mejores estrategias de las M originales. A estas estrategias las llamamos estrategias élite.
7. Generar S estrategias que constituyen los individuos de la población de un AG.
8. Hacer que cada una de las S estrategias compitan con las estrategias élite. La función de medida del individuo depende, entonces, de su desempeño frente a la élite.
9. Del total de las $N+S$ estrategias, elegir las mejores N. Éstas se convierten en la nueva élite.

10. Repetir los pasos 7, 8 y 9 un número predeterminado de veces.

La mejor estrategia es la mejor de la élite al finalizar el algoritmo.

Nótese que, en este algoritmo, la medida de desempeño del individuo depende de una subpoblación que, idealmente, debería ser aquella que estamos buscando. Puesto que esta población ideal nos es desconocida, usamos una aproximación a la misma y la vamos depurando de una manera iterativa. En este sentido, la población élite coevoluciona junto con los individuos de la población misma.

Para ejemplificar el método utilizaremos el problema denominado *dilema del prisionero* (DP). Este problema tiene profundas implicaciones en la teoría de juegos y, de hecho, puede ser visto como una abstracción de una estrategia de adaptación frente a un medio que establece recompensas/castigos

V.2.2 El dilema del prisionero

El problema que vamos a resolver está motivado por la necesidad de determinar una estrategia que nos permita maximizar un beneficio hipotético derivado del comportamiento de otro ente (u *oponente*). Se denomina el dilema del prisionero porque, originalmente, fue descrito como el dilema que dos prisioneros tenían que enfrentar. En este dilema (juego) cada uno de los dos participantes (prisioneros) debe optar entre ayudar (cooperar) o traicionar (abandonar) al otro participante. Este juego es de particular importancia porque la solución dada por la teoría de juegos es evitar la cooperación (llamada abandono) en tanto que la repetición real del juego lleva a los jugadores a descubrir los beneficios de la mutua cooperación.

En el dilema del prisionero, cada jugador tiene sólo dos opciones en cada jugada: cooperar (C) o abandonar (A). Hay, por tanto, cuatro posibles resultados en cada jugada: AA, AC, CA o CC. El pago de estos resultados puede ser dado por una tabla como la siguiente:

TABLA T.V.2.A: *Matriz de pago para el dilema del prisionero*.

		Segundo Jugador	
		Coopera	Abandona
Primer Jugador	Coopera	+3, +3	0, +5
	Abandona	+5, 0	+1, +1

Supondremos consistentemente, que el segundo jugador es nuestro “oponente”. Es decir, accionamos antes que el segundo jugador lo haya hecho. Esto significa que en un par CA, por ejemplo, “él” abandona y “nosotros” cooperamos. En tal caso “él” gana +5 unidades mientras “nosotros” no ganamos ninguna.

La solución minimax dada por la teoría de juegos minimiza el daño máximo que el oponente puede infringir. Éste se determina comparando el máximo daño bajo cooperación contra el máximo daño bajo abandono. Si el primer jugador coopera (C,-), el máximo daño ocurre cuando el segundo jugador hace la respuesta A, dando un pago de 0 para el primer jugador. Si el primer jugador abandona, el máximo daño ocurre, nuevamente, cuando el jugador 2 hace la respuesta A. Ahora el pago al jugador 1 es +1. Así, el primer jugador sufre un daño mínimo abandonando siempre. Este razonamiento es simétrico y así (A,A) es la solución minimax. Sin embargo, de la tabla es evidente que ambos jugadores pueden mejorar cooperando todo el tiempo. Entonces ambos ganan +3 consistentemente; mucho mejor que la solución minimax.

El juego iterado del dilema del prisionero (DPI) consiste en encontrar la mejor estrategia dado que llevamos registro de las últimas *n* jugadas. Por ejemplo, si la secuencia de los últimos 3 juegos ha sido AA, CC y CC, ¿cuál debe ser nuestro próximo movimiento? Claramente, si registramos 3 juegos, hay 2^6 posibles historias. Una estrategia debe especificar para cada una de las historias el movimiento que el jugador debe realizar. Una estrategia particularmente interesante es la llamada *uno-por-otro (tit-for-tat)*. En ésta el jugador empieza cooperando. Después repite el último movimiento del oponente. *Uno-por-otro* ha demostrado ser muy eficiente. Podemos indizar las diferentes historias si consideramos cada "C" como un "1" y cada "A" como un "0". Entonces, la historia AAAAAA recibe el índice 0; igualmente, la historia AACCCC recibe el índice 15, etc. Cualquier estrategia puede ser representada por una cadena de 64 bits.

Índice → 1	0	1	2	3	...	63
Acción → 2	A	C	A	C	...	C

↑ E S T R A T E G I A

En el ejemplo, por tanto, la estrategia especifica A cuando la historia pasada ha sido AAAAAA (0); C cuando la historia ha sido AAAAAC (1), etcétera.

Aún para $n = 3$ el número de estrategias posibles es enorme ($2^{64} \approx 16x10^{18}$). Podemos pensar en un jugador que juega el dilema del prisionero empezando con un conjunto inicial de posibles estrategias que va a probar en contra de su oponente. Podemos también pensar en cada estrategia como un conjunto de reglas de estímulo-respuesta. La historia inmediata anterior es el estímulo que determina la respuesta. El juego usual se estipula como la maximización de la ganancia derivada de una serie de jugadas, en vez de una jugada aislada. Además, varios juegos se efectúan entre una estrategia candidato bajo prueba y un conjunto de estrategias reputadamente buenas que se convierten en la base para determinar el desempeño del candidato. A esto se le llama el dilema del

prisionero iterado (DPI). Para una mayor discusión de las muchas implicaciones del DPI, el lector puede consultar [1], [2], [3].

Para resolver el DPI debemos atacar los siguientes problemas:

i) Encontrar una codificación adecuada.
ii) Encontrar las estrategias básicas (élite inicial).
iii) Optimizar el desempeño.

Encontrar la codificación adecuada es simple. Dado un número p de jugadas previas consideradas, es fácil ver que el número de posibles historias es $H = 2^{2p}$. Por lo tanto, podemos representar cualquier estrategia dada con una cadena binaria de H bits de largo, en donde un *0* representa abandono y un *1* representa cooperación.

Encontrar las estrategias básicas es un poco más difícil. Axelrod [4] convocó a un conjunto de individuos y los invitó a participar en la solución del problema con la presentación de algún programa que pretendiera resolver el DPI. Los participantes respondieron con programas de diversas índoles: desde aquellos que elegían las estrategias de manera aleatoria hasta programas que utilizaban inferencia bayesiana. Una vez que los tuvo en su poder, los hizo competir entre sí y eligió aquellos que se desempeñaron mejor, como los elementos del conjunto básico de estrategias. Nosotros deseamos sistematizar el proceso de Axelrod. Es decir, queremos prescindir del elemento humano. Debemos probar a nuestros candidatos contra un conjunto de estrategias que, supuestamente, se desempeñan de manera superior. Pero, ¿cómo sabemos cuáles son éstas? Si conociéramos a los mejores individuos (estrategias) ya habríamos resuelto el problema. Para evitar este aparente círculo vicioso, aplicamos el siguiente algoritmo:

V.2.3 Algoritmo DPI

a) Generar un conjunto S de m[11] estrategias aleatoriamente.
b) **Desde** i:=1 hasta 6 • 6m
    Elegir al azar dos estrategias s_1, s_2 de S.
    Confrontar s_1 contra s_2 un número predefinido de veces.[12]
    Registrar los resultados tanto de s_1 como de s_2.
  findesde
c) De las S estrategias, elegir las mejores t,[13] es decir, aquellas cuyo comportamiento promedio ha sido el mejor. Tomamos entonces ese conjunto de estra-

[11] El número real es 6 • t (en donde t representa el número de funciones en contra de las cuales las soluciones candidatos van a ser probadas).
[12] Convencionalmente confrontamos las estrategias *50 + 2* (jugadas previas) veces.
[13] Dentro de las mejores t se inserta la estrategia *uno-por-otro*.

tegias (*T*) como la base contra las cuales deberemos de probar a los individuos del algoritmo genético.

d) Generar una población inicial aleatoria de *n* estrategias candidato (*C*).

e) **mientras** criterio de convergencia no sea alcanzado **hacer**

f) **Desde** *i:=1* **hasta** *n*

Elegir la estrategia c_i de *C*.

Desde *j:=1* **hasta** *t*

Probar c_i contra s_j

findesde

findesde

El ajuste de cualquier individuo es el desempeño promedio de tal individuo vs. aquellos en *T*.

g) De los *n* individuos en la población más los *t* individuos en el conjunto base, elegir a los mejores *t*. Éstos constituirán el conjunto base para la siguiente generación.

finmientras

Fin del algoritmo DPI

Tratamos, pues, de aumentar la fuerza del conjunto *T* eligiendo las estrategias globalmente mejores para conformar *T*. Éste evoluciona dinámicamente junto con los individuos de la población. Aquí, por tanto, tenemos el ejemplo de un AG que coevoluciona.

La metodología ejemplificada por el ejemplo anterior puede ser generalizada a cualquier problema en el cual la medida de desempeño del individuo (es decir, la medida de bondad de la solución propuesta) dependa de posibles soluciones alternas. Por ejemplo, si deseamos encontrar la mejor forma de motivar al personal de un grupo de trabajo, o si deseamos seleccionar la mejor manera de repartir los pedidos entre un grupo de proveedores constantes, o si deseamos determinar la mejor forma de repartir los recursos entre un grupo de fabricantes. En todos estos casos la mejor alternativa depende de un sistema de comportamiento dinámico cuyas características van a ser modificadas por nuestra propia decisión.

V.3 AUTÓMATAS GENÉTICOS

Originalmente, la *programación evolutiva* introducida por Fogel [5] se ofreció como un intento de crear inteligencia artificial. El enfoque era el de evolucionar máquinas de estados finitos (MEF) para predecir eventos con base en observaciones previas. Una MEF es una máquina abstracta que transforma un conjunto de símbolos de entrada en una secuencia de símbolos de salida. La transformación depende de un conjunto finito de estados y un conjunto finito de reglas de

transición. El desempeño de una MEF con respecto a su entorno puede ser medido, entonces, con base en las capacidades predictivas de la máquina, es decir, comparando cada símbolo de salida con el siguiente símbolo de entrada y midiendo el valor de la predicción con alguna función de pago. En contraste con los AGs, la representación se basa directamente en vectores de valores reales cuando se trata con problemas de optimización de parámetros continuos de la forma general

$$f : M \subseteq R^n \rightarrow R$$

En el caso que nos ocupa, seguimos el mismo enfoque mencionado en los párrafos precedentes, con las siguientes particularidades:

a) Usamos máquinas de Turing (MT) en vez de máquinas de estados finitos.
b) Utilizamos una codificación genética de manera que los operadores de selección, cruzamiento y mutación son directamente aplicables.

La razón por la cual optamos por MTs en vez de MEFs es simple: las MTs constituyen los autómatas con capacidades más amplias de cómputo, como lo demostró A. Turing en su famoso trabajo [6].

V.3.1 Máquinas de Turing

Expresado en términos simples, las MTs son dispositivos diseñados para mostrar los elementos más esenciales de un procedimiento claramente definido, a través del cual uno puede expresar la potencia computacional de todas las funciones recursivas parciales. Además, si tenemos cuidado en la forma de definir una MT, es inmediato establecer una correspondencia uno a uno entre las funciones básicas, la composición, la recursión primitiva y la minimalización y la correspondiente MT [7]. Esquemáticamente, una MT puede ser representada como se muestra en la figura F.V.3.A.

En tal dispositivo, existe una cadena de símbolos denotados en la figura por S_1, S_2,..., S_5. Estos símbolos se ubican en una cinta que se extiende hacia $-\infty$ y $+\infty$ a la izquierda y derecha, respectivamente. Una cabeza de lectura/escritura se mueve analizando los símbolos en la cinta y tomando una de las siguientes acciones:

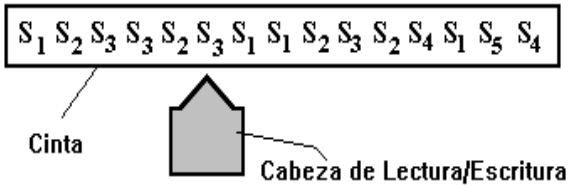


FIGURA F.V.3.A: *Una máquina de Turing.*

a) Escribe uno de los símbolos del alfabeto en la cinta.
b) Se mueve una posición a derecha o izquierda (o no se mueve) de la posición actual.

Esto se hace un paso a la vez cuando la MT pasa de un estado a otro. Por estado significamos, precisamente, la configuración de la máquina. La forma en la que la MT pasa de estado a estado, está definida por un programa (o unidad) de control tal como el que se muestra en la figura F.V.3.B.

Estado Actual	Entrada = 0			Entrada = 1		
	Nuevo Símbolo	Mov. de la Cabeza	Sig. Estado	Nuevo Símbolo	Mov. de la Cabeza	Sig. Estado
Q1	1	DER.	Q2	1	IZQ.	Q3
Q2	0	IZQ.	Q1	0	IZQ.	Q4
Q3	1	IZQ.	Q1	1	IZQ.	PARO
Q4	1	IZQ.	Q2	1	DER.	Q5
Q5	0	DER.	Q4	0	DER.	Q2

FIGURA F.V.3.B: *La Unidad de control de una máquina de Turing.*

La MT particular cuya unidad de control se muestra, se presume que opera con un alfabeto binario en una cinta que está, inicialmente, llena de 0's. Por ello hay dos columnas: la primera corresponde a las acciones que la máquina debe efectuar cuando el símbolo bajo la cabeza es *0*. Igualmente, la segunda columna corresponde a las acciones que debe tomar cuando el símbolo leído sea *1*. Note que en el estado Q3, cuando se presenta un *1* en la cinta, lleva a un estado etiquetado PARO. Este estado especial significa que la MT debe detener toda actividad. Es posible que la MT en cuestión entre a este estado de paro desde varios estados o de ninguno. Un problema interesante es encontrar cuál MT, dado un número de estados fijo, es capaz de escribir el número mayor de 1's en una cinta

inicialmente llena de 0's.[14] Se invita al lector a seguir el comportamiento de la MT de la figura anterior.

Es común hablar de una MT cuando uno se refiere a una unidad de control. En verdad, el trabajo interno de cualquier MT depende exclusivamente de dicha unidad. A menos que se indique, nos adheriremos a esa convención y hablaremos simplemente de la MT cuando nos refiramos a su unidad de control.

Debe resultar claro, de la discusión antecedente, que ninguna MT es físicamente realizable ya que su cinta es infinita. Lo más que podemos hacer es simular la MT con una cinta finita (aunque arbitrariamente larga). El interés principal en estos "dispositivos" es que existe una relación biunívoca entre las funciones recursivas parciales y las MTs. Dado lo anterior, es posible afirmar que las MTs son dispositivos de cálculo universales y más generales que los que se obtienen de utilizar máquinas de estados finitos. Esta últimas son un subconjunto de las MTs.

V.3.2 Viabilidad de un autómata genético

Para definir con exactitud el modelo propuesto, hacemos las siguientes afirmaciones de viabilidad:

AFIRMACIÓN 1. Una tabla de transición tal como la que se muestra en la tabla T.V.3.A representa un algoritmo.

TABLA T.V.3.A: *Una tabla de transición.*

Estado Actual	Salida, Movimiento, Siguiente Estado	
	Entrada=0	Salida=0
Q0	1,D,Q1	0,I,Q0
Q1	0,D,Q2	---
Q2	0,D,Q3	1,D,Q3
Q3	1,I,Q4	1,D,Q3
Q4	0,I,Q5	1,I,Q4
Q5	0,D,Q6	1,I,Q5
Q6	0,D,Q7	0,D,Q2
Q7	0,I,Q8	1,D,Q7
Q8	---	0,I,Q9
Q9	1,I,QA	1,I,Q9
QA	1,I,QA	0,D,QB
QB	0,D,QB	0,D,QB

[14] El máximo número de 1's escrito en la cinta de la MT se denomina la productividad. La función que denota la productividad de una MT de *n* estados se simboliza con $\sigma(n)$ y se conoce como la función de Rado.

Esta tabla representa la unidad de control de una MT. La columna de la izquierda representa el estado en el que la MT está. Las dos columnas de la derecha representan la salida, el movimiento de la cabeza y el siguiente estado para las entradas 0 y 1, respectivamente. Las letras *I* y *D* significan izquierda y derecha. El símbolo '-' representa una condición de 'no importa'.

En el algoritmo que se muestra, se supone que la cinta contiene *n* 1's en una cadena ininterrumpida; que el primer símbolo a ser leído es el 1 más a la izquierda; que inicialmente la MT está en el estado Q0. Dado lo anterior, el algoritmo representa una función en *n*. Ninguna MEF puede reproducir este comportamiento para una *n* arbitraria.

AFIRMACIÓN 2. Tablas tales como las que se incluyen en la tabla T.V.3.A pueden ser interpretadas como una secuencia de elementos independientes a los que denominaremos "genes".

Para ver cómo es posible esto, haremos una codificación binaria de la tabla de transición asignando un número arbitrario a cada uno de estos estados. En la tabla T.V.3.B mostramos tal codificación binaria. Los encabezados de la columna E, M y SE significan Entrada, Movimiento (de la cabeza de la MT) y Siguiente Estado, respectivamente.

TABLA T.V.3.B: *Una MT como una cadena de cuartetas.*

```
1000,0101,0000,0000,10--,----,0000,1110,0011
1101,0010,0011,0101,0111,0100,0001,1011,0101
0001,1100,0010,0110,0010,0111,----,--01,1001
1110,1011,1001,1110,1000,1011,0010,1111,1011
```

Ahora agrupamos los bits en cuartetas, como se muestra en la tabla T.V.3.C.

TABLA T.V.3.C: *Codificación hexadecimal de una MT.*

```
8,5,0,0,8,0,0,E,3,D,2,3,5,7,4,1,B,5,1,C,2,6,2,7,0,1,5,E,B,9,E,8,B,2,F,B
```

Finalmente, reemplazamos los 'no importa' por 0's y escribimos cada cuarteta como un dígito hexadecimal como se muestra en la tabla T.V.3.D.

Definimos O(P) como el orden de cualquier autómata en una población P de tamaño *n*. En el ejemplo, O(P)=144.

TABLA T.V.3.D: *Codificación binaria de una MT.*

E M SE	E M SE
1 0 0001	0 1 0000
0 0 0010	---
0 0 0011	1 0 0011
1 1 0100	1 0 0011
0 1 0101	1 1 0100
0 0 0110	1 1 0101
0 0 0111	0 0 0010
0 1 1000	1 0 0111
---	0 1 1001
1 1 1010	1 1 1001
1 1 1010	0 0 1011
0 0 1011	0 0 1011

AFIRMACIÓN 3. De un conjunto de autómatas con características aleatorias es posible identificar (extraer) aquellos genes que determinan la MT cuyas características de comportamiento son óptimas bajo una norma dada.

La afirmación 1 se sigue directamente de la tesis de Turing. La afirmación 2 se demuestra simplemente generalizando el esquema de codificación delineado arriba. La afirmación 3 se refiere al modelo que usamos. Se describe en lo siguiente.

a) Una población inicial (de n MTs) se genera. n se elige arbitrariamente (en relación con este punto, véase [8]).
b) Una cadena fuente inicial se establece. Ésta constituye la entrada binaria al autómata al inicio de los cálculos.
c) Se establece una cadena objetivo. Ésta es una secuencia binaria que el autómata debe predecir a partir de una secuencia binaria de entrada.
d) Para cada autómata en la población, se ejecuta un número prefijado de pasos. Naturalmente, las acciones tomadas en cada paso son función de su estructura.
e) El autómata debe escribir el t-ésimo símbolo correspondiente al símbolo t de la entrada. Cada MT es calificada de acuerdo con la cantidad de símbolos que ha sido capaz de predecir.
f) Los autómatas se ordenan de mejor a peor de acuerdo con su función de desempeño. La mejor MT recibe el número 1 y la peor recibe el número n. Dado este ordenamiento, definimos un conjunto de $\eta=n/2$ parejas. La primera pareja consiste de los individuos 1 y n; la segunda corresponde a los individuos 2 y $n-1$, etc. En general, la pareja i consiste de los individuos i y n-

i+1. En cada pareja designamos al individuo con mejor desempeño como el origen; el otro como el destino.

g) Con probabilidad *c* (para cada una de las η parejas), se lleva a cabo el siguiente procedimiento:

1) Se eligen al azar dos puntos de cruzamiento. El primero de ellos corresponde al origen; el segundo al destino.
2) Una cadena de longitud *O(P)/2* se elige en el punto de origen; otra se elige en el punto de destino. Las cadenas genéticas se consideran como anillos. Es decir, el bit más a la derecha de la cadena se considera contiguo al bit más a la izquierda.
3) Se efectúa el cruzamiento entre fuente y destino como se muestra en la figura F.V.3.C.

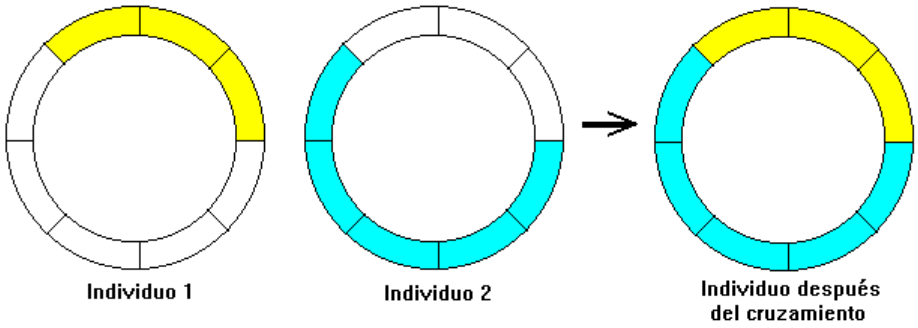


FIGURA F.V.3.C: *Cruzamiento anular*.

El siguiente es un paso crítico en el esquema propuesto. Tradicionalmente, en este punto a algunos de los individuos de la población se les permite sobrevivir; tal vez un individuo pasa varias copias de sí mismo a la siguiente población, antes del cruzamiento [9]. Entonces, los individuos sobrevivientes son elegidos aleatoriamente y cruzados. Aquí, «todos los individuos elegidos como fuente son cruzados con los individuos elegidos como destino».

h) Se induce una mutación con probabilidad *m*.

Los pasos (a) a (h) se repiten un número fijo de veces o hasta que se encuentra un predictor perfecto. Tal predictor es lo que denominamos un autómata genético.

Nuestra contención es que la población evoluciona positivamente. Es decir, que los autómatas pueden predecir más atinadamente en la generación *g+1* que

en la generación *g*. Por tanto, el proceso descrito garantiza que el autómata aprenda.

V.3.3 Algunas consideraciones en referencia al modelo planteado

El modelo determinístico de cruzamiento fue diseñado para, específicamente, evitar el problema que nace debido a la predominancia de un individuo sobresaliente (para la discusión de este fenómeno vea, por ejemplo, a [9]). Cruzando el mejor y el peor individuos la población evoluciona, en vez de un individuo o un subconjunto de ellos, de una manera que evita que el fondo genético se sature por las características de un solo individuo. Además, el esquema de cruzamiento elegido tiende a enriquecer al fondo, aunque favorece la persistencia de los mejores esquemas en función del desempeño de la población y no del desempeño de un individuo.

Por otro lado, con cruzamiento anillado podemos evitar algunos problemas de dependencia posicional de la codificación.

Otro punto a considerar es la codificación de la MT. Arbitrariamente hemos seleccionado una representación de las tablas, de la forma ilustrada. Otros tipos de codificación son ciertamente posibles.

Una pregunta abierta es la referente a las ventajas/desventajas de esta forma de codificación.

V.3.3.1 Determinación de la validez del modelo

Para determinar la validez de la estrategia propuesta se efectuaron una serie de experimentos [10] que, brevemente, se mencionan a continuación.

EXPERIMENTO 1. El primer conjunto de corridas se enfocaron a determinar si el modelo propuesto rinde los resultados esperados en términos de convergencia.

Para una cadena de longitud 40, se compararon las capacidades predictivas de poblaciones aleatorias y poblaciones genéticas. Para cadenas cortas, la aleatoriedad juega un papel importante. Sin embargo, una vez que las condiciones de predicción son más restrictivas, marcadamente los procesos genéticos superan a los procesos aleatorios. Los resultados se muestran en la figura F.V.3.D.

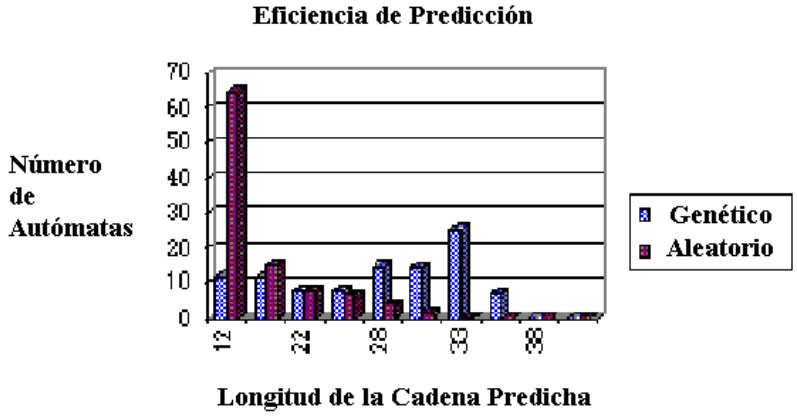


FIGURA F.V.3.D: *Eficiencia de predicción contrastada entre un proceso genético y un proceso aleatorio.*

EXPERIMENTO 2. Se simularon 8 poblaciones de 200 generaciones cada una y 2 poblaciones de 800 generaciones cada una. Se pudo observar que hay un efecto acumulativo de convergencia. Mientras más largas sean las cadenas a predecir, la población de 800 generaciones fue superior. Esto confirma la propiedad de memoria en los procesos de aprendizaje (véase figura F.V.3.E).

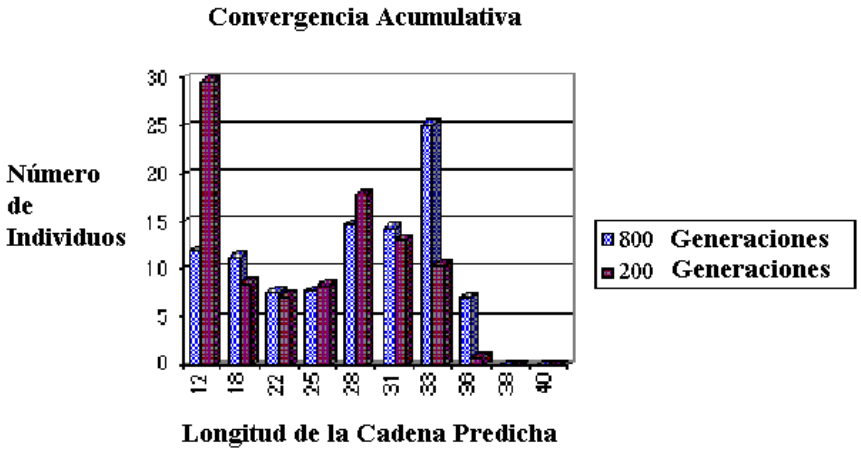


FIGURA F.V.3.E: *Memoria de los procesos genéticos.*

En los casos mencionados, los mejores resultados son proporcionales a la longitud de las corridas. Hipotéticamente la curva de aprendizaje pudiera ser no monotónicamente incremental. Lo que se mostró es que tal no es el caso, el proceso de aprendizaje es monotónicamente creciente. Como conclusión, el modelo propuesto es viable. Esto, a pesar de la codificación arbitraria.

V.3.3.2 Generalidad del modelo

Las siguientes ejecuciones se diseñaron para determinar si los resultados anteriores son aplicables en general. Queremos, pues, determinar si las características de aprendizaje se aplican en todos los casos.

EXPERIMENTO 3. Se define el coeficiente de información (CI) como la capacidad predictiva para un conjunto de MTs, dadas cadenas origen y destino arbitrarias. Se calculó el CI para 600 generaciones. La capacidad predictiva aumenta constantemente a medida que el autómata genético evoluciona en el tiempo. Se simularon 90 procesos genéticos de 600 generaciones cada uno. En cada uno de éstos la capacidad predictiva se calculó. Se efectuó un análisis de varianza. De este análisis es posible concluir que todos los procesos genéticos convergen a capacidades predictivas similares con probabilidad 0.97 o mejor. Es decir, independientemente de la cadena fuente y destino los procesos dieron origen a autómatas genéticos con capacidades predictivas similares.

El análisis de varianza mostró que, sin importar las condiciones iniciales, los algoritmos se desempeñan de manera que podemos afirmar que provienen de la misma población. Esta es una conclusión importante ya que nos permite esperar capacidades predictivas similares, sin importar la función de entrada cuyo comportamiento tratamos de dilucidar. Los datos anteriores se ilustran en la figura F.V.3.F.

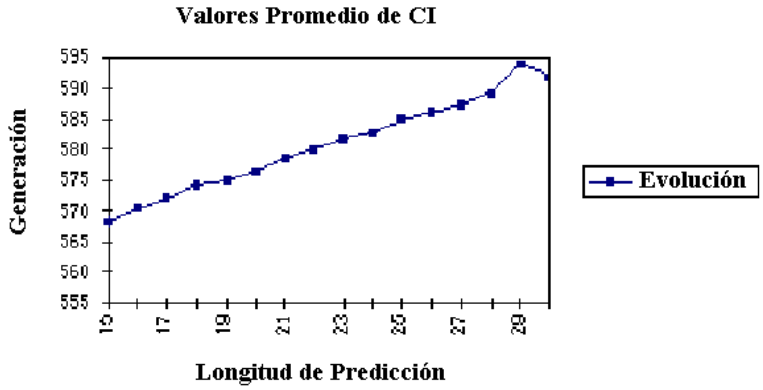


FIGURA F.V.3.F: *Capacidad predictiva de un autómata genético*.

V.3.3.3 Probabilidades de cruzamiento

Habiendo determinado que el modelo propuesto tiene características de desempeño interesantes nos importa su eficiencia. Uno de los asuntos que nos ocupa es si P_c afecta el desempeño del algoritmo y, de ser así, cómo.

EXPERIMENTO 4. Se llevaron al cabo 20 procesos genéticos de 500 generaciones cada uno. Se probaron distintas probabilidades desde un mínimo de 0.05 hasta una P_c de 1. Probabilidades bajas arrojaron convergencias más lentas. Las probabilidades óptimas se encontraron en 0.55 y 0.85. Un valor alto de P_c no necesariamente constituye la mejor elección.

Desde un punto de vista práctico, la correcta elección de P_c es importante. En términos de *flops*, una diferencia de 30% es significativa. La velocidad de convergencia se ilustra en la figura F.V.3.G.

V.3.3.4 Coeficiente de correlación predictiva

Nos interesa determinar qué tan predecible es el proceso de aprendizaje del modelo. Buscamos encontrar regularidades (si existen) en el proceso de aprendizaje. Para ello establecimos la correlación entre las predicciones del mejor autómata el tiempo *t* vs. el desempeño del mejor autómata al tiempo *t+1*.

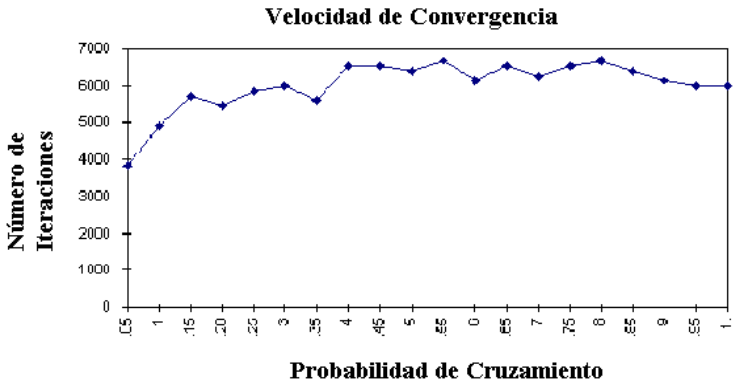


FIGURA F.V.3.G: *Velocidad de convergencia en función de las probabilidades de cruzamiento de un autómata genético*.

EXPERIMENTO 5. Del experimento se desprende el hecho de que la capacidad predictiva y los valores observados exhiben una estrecha correlación. A esta correlación la denominamos "coeficiente de correlación predictiva".

En esta simulación, se codificó el mejor autómata de cada era (definida como un conjunto de 50 generaciones consecutivas) y encontramos polinomios de 8^o grado en dos variables independientes. Es decir, encontramos una relación de la forma

$$G = f (I, T)$$

en donde

G = El mejor autómata
I = información fuente
T = etapa (era) del proceso genético

Con esta función encontramos los errores predictivos de los valores extrapolados de G_{k-1} (E_{pred}) al tiempo t_k. Igualmente, los errores de ajuste para G_k en t_k (E_{fit}). El coeficiente de correlación predictiva es, entonces,

$$CCP = E_{pred} / E_{fit}$$

Los *CCP*s se obtuvieron para 50 eras (2500 generaciones) y se efectuó un análisis de varianza. De este análisis es posible concluir que, con probabilidad 0.99, es posible predecir el comportamiento para t_{k+1} dado que conocemos el comportamiento del proceso en t_k. El comportamiento señalado se ilustra en la figura F.V.3.H.

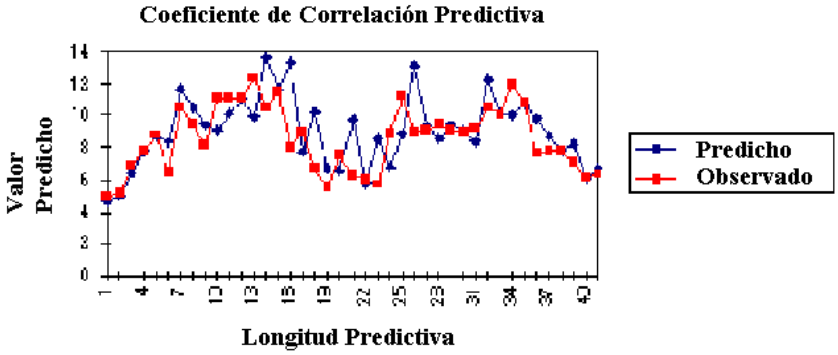


FIGURA F.V.3.H: *Predictibilidad de la evolución de un autómata genético.*

V.4 SISTEMAS CLASIFICADORES EVOLUTIVOS

Durante las últimas dos décadas ha habido un interés creciente en la aplicación de técnicas de programación al aprendizaje de máquinas basado en evolución (AMBE) [12]. Esto está basado en la idea de que los cromosomas, al representar conocimiento, sean tratadas como datos a ser manipulados por los operadores genéticos y, al mismo tiempo, como código ejecutable que sea utilizable para desarrollar alguna tarea. En general, en la comunidad de computación evolutiva hay dos enfoques que compiten en la solución del problema:

Una forma natural de proceder es desarrollar todo el conjunto de reglas como una cadena (un individuo), mantener una población de conjuntos de reglas candidatas y usar los operadores genéticos para producir nuevas generaciones de conjuntos de reglas. Históricamente este fue el enfoque seguido por De Jong y sus estudiantes en la Universidad de Pittsburg. Ello dio origen a la frase "el enfoque Pitt".

Otra forma (desarrollada por Holland por la misma época) es la de aplicar un modelo de conocimiento (sistemas clasificadores) en la cual los miembros de la población son reglas individuales y un conjunto de reglas está representado por toda la población. A este enfoque se le denominó "el enfoque Michigan" (por la universidad en donde Holland trabajaba en esas fechas).

En este apartado hacemos una descripción del enfoque Michigan y los denominados Sistemas Clasificadores Evolutivos, a los que da origen. En el proceso de aprendizaje, uno de los enfoques más interesantes es el de los sistemas adaptivos complejos. En éstos los sistemas aprenden del medio ambiente de manera automática. La descripción que sigue se ha basado en los comentarios de Goldberg [9].

Un sistema clasificador evolutivo[15] (SCE) consta de tres sustratos:

1. Un sistema de producción paralelo.
2. Un algoritmo de asignación de crédito.
3. Un algoritmo de descubrimiento de clasificadores.

El sistema de producción modela el dominio del problema como conglomerados de reglas de producción altamente estandarizadas, llamadas *clasificadores*, y suministra un ciclo básico de inferencia de pareo[16]-selección-acción con activación paralela de clasificadores. El algoritmo de asignación de crédito evalúa la fuerza de cada clasificador basado en retroalimentación del medio ambiente. Esta fuerza sirve como la medida de la utilidad del clasificador para el SCE y es usado tanto en el proceso de inferencia como en el descubrimiento de nuevos clasificadores. Los algoritmos de descubrimiento de clasificadores son, típicamente, una combinación de algoritmos genéticos y métodos heurísticos. En conjunto, la asignación de crédito y el descubrimiento de clasificadores son las técnicas que dotan al SCE de su capacidad de adaptación, que es lo que permite a este tipo de sistemas de aprendizaje responder a las condiciones cambiantes en el dominio del problema.

V.4.1 Sistema de reglas de producción

En esta metodología en particular, se propone la utilización de reglas de producción de la forma

si *condición* **entonces** *acción*

Las reglas de producción[17] son computacionalmente completas, es decir, usándolas como elementos básicos es posible lograr cualquier operación lógica. Esta es la razón por la cual un gran número de sistemas expertos se fundamentan en este tipo de construcciones.

Que las reglas de producción sean computacionalmente completas se desprende de las siguientes observaciones. La construcción **si** *condición* **entonces** *acción* puede ser reemplazada por *condición* **implica** *acción* o, simbólicamente, por $A \supset B$ (que se lee *A implica B*). La tabla de verdad de la implicación lógica es la mostrada en T.V.4.A.

---

[15] O sistema clasificador con capacidad de aprendizaje.

[16] Entendemos por *parear* un clasificador con una condición al hecho de que la condición sea satisfecha por el clasificador en cuestión.

[17] En psicología se les conoce como reglas de estímulo-respuesta; en inteligencia artificial se conocen como reglas de condición-acción.

TABLA T.V.4.A: *Tabla de verdad de la función de implicación.*

A	B	$A \supset B\ (\overline{A} + B)$
F	F	V
F	V	V
V	F	F
V	V	V

En la tabla anterior, las *F*s significan *falso* mientras que las *V*s significan *verdadero*. Podemos mostrar que la función de implicación es completa si de ella desprendemos los operadores lógicos de negación (*NOT*) y alguno de los operadores de conjunción (*AND*) o de unión (*OR*). Del teorema de De Morgan[18] se sigue que los operadores *NOT* y *AND* o *NOT* y *OR* (tomados así, en pares) son suficientes para expresar cualquier función lógica.

La obtención del operador *NOT* a partir de la equivalencia puede esquematizarse en la siguiente figura (F.V.4.A).

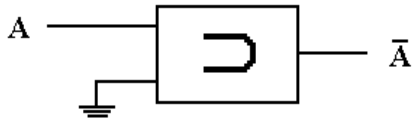


FIGURA F.V.4.A: *Operador de negación.*

De manera análoga, el operador de unión (*OR*) se puede obtener como se muestra en la figura F.V.4.B.

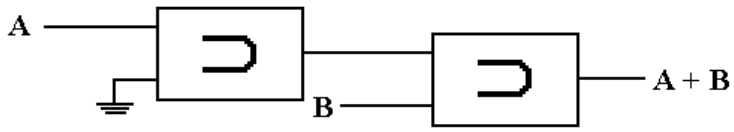


FIGURA F.V.4.B: *Operador de unión.*

[18] Si A y B son variables booleanas, entonces:

$$\overline{A + B} = \overline{A} \bullet \overline{B}$$
$$\overline{A \bullet B} = \overline{A} + \overline{B}$$

Como es claro, una vez obtenidos estos operadores a partir del operador de equivalencia, es trivial extenderlos para de allí obtener cualquier función booleana.

Las reglas de producción deben ser complementadas con un sistema de interacción. Cuando este sistema es provisto, al conjunto de reglas de producción se le dota de una capacidad equivalente a un lenguaje de programación. En un sistema clasificador evolutivo, una regla es capaz de activar a otra u otras reglas como se describirá más adelante. Por el momento, consignaremos que las reglas de producción serán representadas por cadenas de longitud fija. Cada cadena está formada por dos partes. Una parte corresponde a la *condición*; la otra corresponde a la *acción*.

La subcadena de *condición* se define en un alfabeto ternario, de longitud *l*, de manera que *condición* $\in\{0,1,\#\}^l$. Cada *0* ó *1* en la cadena corresponde a un elemento de la codificación de los estímulos del medio ambiente. El símbolo # corresponde a una condición de *no-importa*. De manera análoga, la subcadena de *acción* se define en un alfabeto binario, de manera que *acción* $\in\{0,1\}^l$. Cada uno de los 0's ó 1's corresponde a la codificación de un mensaje que corresponde a una acción.

El estrato de producción del SCE guarda muchas similitudes con los sistemas expertos basados en reglas. En particular, el conocimiento del sistema de producción se encuentra codificado en un conjunto de clasificadores vía un ciclo de pareo-selección-actuación. La diferencia primordial entre los dos tipos de sistemas yace en el mecanismo del sistema de producción para activación paralela de los clasificadores, lo que lo hace un sistema basado en clasificación paralela. Por otra parte, los sistemas expertos son de naturaleza secuencial, permitiendo que sólo una regla se procese a la vez. En una memoria de corto alcance se mantiene una lista de mensajes global que almacena los mensajes internamente generados así como los mensajes de comunicación de entrada/salida del medio ambiente. Un conjunto de detectores y efectores provee la interfaz basada en mensajes con el medio ambiente. Un ejemplo de un detector es un sensor de temperatura, en tanto que un ejemplo de un efector es un brazo robot o una válvula.

Cada clasificador tiene, como ya se mencionó, una sintaxis de la forma **si** *condición* **entonces** *acción* (por ejemplo **si** *temperatura*$>100^oC$ **entonces** *abre la válvula*). Condiciones y acciones son cadenas de longitud fija y de idéntica estructura para todos los clasificadores. Los mensajes son idénticos en estructura excepto que no contienen el símbolo #. Un mensaje en un SCE es, simplemente, una cadena de longitud finita sobre un alfabeto finito. Si nos limitamos al alfabeto binario, entonces:

<Mensaje> ::= { 0, 1}^{l}

Los mensajes en la lista de mensajes pueden satisfacer uno o más clasificadores o reglas de cadena. Un clasificador es una regla de producción con la siguiente sintaxis:

<Clasificador> ::= <Condición> : <Mensaje>

La condición es un dispositivo de reconocimiento de patrones en donde un carácter de *no-importa* (#) se agrega al alfabeto subyacente.

<Condición> ::= {0, 1, #} [1]

Así, la condición es satisfecha por un mensaje si en cada posición un 0 en la condición caza con un 0 en el mensaje, un 1 caza con un 1 ó un # caza con un 0 ó un 1. Por ejemplo, a la condición de cuatro posiciones #11# la satisface el mensaje 1110, pero no la satisface el mensaje 0010.

El clasificador antes mencionado podría estructurarse de la siguiente manera:

Atributo a verificar	Valor a verificar	Acción	Objeto a afectar
Temperatura	^{o} Centígrados	Abrir	Válvula
0010	01100100	0010	0001
	Condición Acción <Mensaje>		
	001001100100:00100001		
	00####100100:00100001		
	##1001100100:00100001		

FIGURA F.V.4.C: *Ejemplo de clasificadores.*

Una vez que la condición de un clasificador es satisfecha, el clasificador se convierte en un candidato susceptible de poner su mensaje en la lista de mensajes en el siguiente paso.

Un sistema de producción de un SCE consiste de una lista de clasificadores, una lista de mensajes, un conjunto de detectores, un conjunto de efectores y un mecanismo de retroalimentación (figura F.V.4.D). También se muestran los sustratos de asignación de crédito y el de descubrimiento de nuevos clasificadores.

El lazo básico de ejecución que gobierna la interacción entre estos componentes consiste de seis pasos para un ciclo de ejecución:

1. Cualquier mensaje de los detectores del medio ambiente se agrega a la lista de mensajes actuales.
2. Los contenidos de la lista de mensajes se comparan con las condiciones de todos los clasificadores.
3. Aquellos clasificadores cuyas condiciones fueron satisfechas, compiten de acuerdo a su fuerza por el derecho de agregar sus mensajes a la lista de men-

sajes, de tal manera que aquellos con mayor fuerza tengan mayor probabilidad de ganar.
4. Los ganadores de la competencia crean nuevos mensajes basados en sus acciones y los mensajes satisfechos.
5. Los nuevos mensajes se agregan a la lista de mensajes.
6. Los efectores llevan a cabo acciones especificadas en la lista de mensajes.

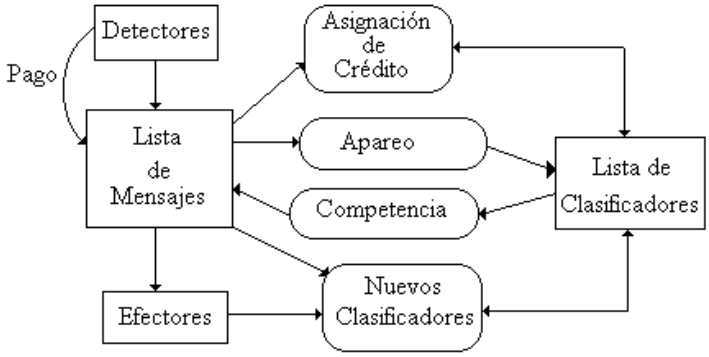


FIGURA F.V.4.D: *Diagrama de bloques de un sistema clasificador evolutivo.*

V.4.2 Asignación de crédito

El propósito de la asignación de crédito en un SCE es distribuir la retroalimentación proveniente del medio ambiente en la forma de un valor de reforzamiento escalar, tal que los clasificadores benéficos sean recompensados y los clasificadores lesivos sean penalizados respecto al resultado buscado.

El algoritmo de Holland que denominaremos algoritmo de cascada,[19] es un mecanismo que puede resolver el problema de asignación de crédito en un SCE. En la figura F.V.4.E se ilustra el proceso del algoritmo de cascada para un sistema de clasificadores sobre un alfabeto binario.

Este algoritmo se ejemplifica con los siguientes clasificadores:

01##:0000 00#0:1100 11##:1000 ##00:0001

[19] Algoritmo de *Bucket Brigade*.

	t=0				
	CLASIFICADOR	FUERZA	PAR	APUESTA	MENSAJES
1	01##:0000	200	MA	20	0000
2	00#0:1100	200			
3	11##:1000	200			
4	##00:0001	200			
Medio Ambiente		0 0111			

	t=1				
	CLASIFICADOR	FUERZA	PAR	APUESTA	MENSAJES
1	01##:0000	180			
2	00#0:1100	200	1	20	1100
3	11##:1000	200			
4	##00:0001	200	1	20	0001
Medio Ambiente		20			

	t=2				
	CLASIFICADOR	FUERZA	PAR	APUESTA	MENSAJES
1	01##:0000	220			
2	00#0:1100	180			
3	11##:1000	200	2	20	1000
4	##00:0001	180	2	18	0001
Medio Ambiente		20			

	t=3				
	CLASIFICADOR	FUERZA	PAR	APUESTA	MENSAJES
1	01##:0000	220			
2	00#0:1100	218			
3	11##:1000	180			
4	##00:0001	162	3	16	0001
Medio Ambiente		20			

	t=4				
	CLASIFICADOR	FUERZA	PAR	APUESTA	ACCION
1	01##:0000	220			
2	00#0:1100	218			
3	11##:1000	196			
4	##00:0001	146			0001
Medio Ambiente		20			

	t=5			
	CLASIFICADOR		RECOMPENSA	FUERZA
1	01##:0000			220
2	00#0:1100			218
3	11##:1000			196
4	##00:0001		50	196
Medio Ambiente		20		

FIGURA F.V.4.E: *Algoritmo de cascada.*

Cuando los clasificadores se parean (satisfacen una condición) no ponen sus mensajes directamente. En vez de esto, cada clasificador pareado puede participar en una subasta. Para ello, cada clasificador mantiene un registro de su valor neto, denominado su *fuerza*. Entonces, cada clasificador hace una apuesta proporcional a su fuerza. Un clasificador pareado y activado paga su apuesta a aquellos clasificadores responsables de haber mandado los mensajes que satisficieron la condición del clasificador que apostó.

En el ejemplo, todos los clasificadores inician con una fuerza de 200. Las apuestas que someten a la subasta dependen de un parámetro de apuesta denominado C_{ap} que, en este caso, cumple con:

$$C_{ap} = 0.1$$

En la práctica, el valor de la apuesta no es directamente proporcional a la *fuerza* (*F*) del clasificador, como se mostró en el ejemplo. Al producto $F \bullet C_{ap}$ se le suma un término de ruido que, típicamente, se toma de una distribución normal con $\mu=0$ y una desviación estándar σ_{ap}. De esta manera la apuesta efectiva (*AE*) se calcula como

$$AE = F \bullet C_{ap} + N(0, \sigma_{ap})$$

De manera complementaria, se manejan dos tipos de impuestos: uno que se activa en cada ciclo del algoritmo (impuesto de vida) y otro que se activa cada vez que el clasificador hace una apuesta (impuesto de apuesta). El primero tiene como objetivo acelerar la desaparición de clasificadores poco activos. El segundo tiene como objetivo evitar una sobrecarga que polarice a la población excesivamente hacia un solo tipo de reglas de producción. El esquema de taxación más simple consiste de un producto proporcional a la fuerza del clasificador. Para ello es necesario definir dos constantes de impuestos, llamadas C_{ivi} y C_{iap}, correspondientes al impuesto de vida y de apuesta, respectivamente. De esta manera, la fuerza de cada clasificador se ve afectada de la forma:

$$F \leftarrow F \bullet C_{ivi} \bullet C_{iap}$$

en donde

$$0 \leq C_{ivi} \; y \; C_{iap} \leq 1$$

Una última consideración es la siguiente. Cuando dos clasificadores satisfacen una condición, aquel que es más específico lleva más información que el que lo es menos. Por ejemplo, los clasificadores ##00, 0#00 y 0100 satisfacen todos a la condición 0100. Pero el último es más específico que el segundo y éste es más específico que el primero. Intuitivamente, en igualdad de circunstancias, un agente adaptivo debe preferir reglas que usan más información acerca de una situación. La forma más simple de implantar la preferencia es modificar la mecánica del proceso de apuesta haciendo que ésta sea proporcional al producto

de la fuerza y la especificidad. Una medida simple de especificidad es el número de posiciones no ambiguas de la condición. De manera equivalente, para una cadena de longitud *l*, especificidad = *número de "#" en la condición*. De esta manera, si ya sea la fuerza o la especificidad son cercanas a cero, la apuesta será cercana a cero; solamente si ambas son grandes la apuesta lo será. La apuesta, entonces, debe calcularse como el producto de una función linear de la especificidad del clasificador y la fuerza del mismo. Una posible función de la especificidad (ESP) es de la forma

$$ESP = apuesta_1 + apuesta_2 \bullet especificidad$$

en donde $apuesta_1$ *y* $apuesta_2$ son parámetros del sistema. Modificando los valores de estos dos parámetros es posible investigar diferentes estructuras de apuesta.

Finalmente, la apuesta efectiva está dada por

$$AE = F \bullet C_{ap} \bullet ESP + N(0, \sigma_{ap})$$

V.4.3 Algoritmo de descubrimiento de clasificadores

El último sustrato de un SCE es el que nos permite la búsqueda de nuevos clasificadores, que no se encuentran en el conjunto inicial. Como ya se señaló, una alternativa para llevar a cabo esa búsqueda es vía un algoritmo genético.

Con base en lo tratado en el capítulo 2, debe resultar evidente que una de las razones por las cuales se ha establecido una sintaxis como la propuesta en los párrafos precedentes, es que las cadenas de longitud fija son directamente susceptibles de ser afectadas por los operadores genéticos discutidos en los capítulos 2 y 3.

En un SCE, un AG es invocado periódicamente para crear nuevos clasificadores. El ciclo básico es (figura F.V.4.F):

1. De la lista de clasificadores, elija aleatoriamente pares de clasificadores padres de manera que los clasificadores más fuertes tengan una probabilidad de selección más alta.
2. Se crean nuevos clasificadores aplicando operadores genéticos a los padres.
3. Se eligen al azar aquellos clasificadores que van a ser reemplazados por los clasificadores recientemente creados, de manera que los clasificadores de menor fuerza tienen mayor probabilidad de ser elegidos.

El ciclo básico corresponde al algoritmo genético simple. Obviamente, es posible usar algoritmos más sofisticados, tales como el algoritmo genético ecléctico mencionado en el capítulo 4.

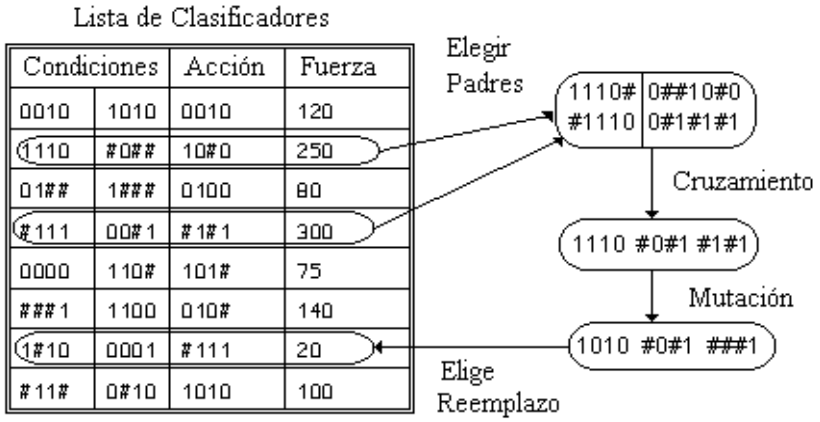


FIGURA F.V.4.F: *Ejemplo del ciclo de un algoritmo genético.*

V.4.4 Jerarquías por omisión[20]

El conjunto de reglas forma un modelo simple del medio ambiente. Es un modelo aparentemente no resuelto porque sus reglas pueden ser contradictorias al resolverse. Sin embargo, una observación más cuidadosa revela que hay, más bien, un proceso simbiótico entre reglas. Cuando existen dos reglas, digamos r_1 y r_2 en donde r_1 es más general que r_2, la regla más general se convierte en una especie de *default*[21] que será usado cuando no se disponga de información de detalle. Sin embargo, si esta regla fuese siempre invocada, el sistema entraría en un estado de amarre que lo inmovilizaría. La regla más específica, r_2, por otra parte, induce una acción correcta en presencia de mayor información. Provee una excepción a la regla *default* y, puesto que es más específica, supera al *default* cuando existen condiciones más específicas. El siguiente argumento revela la simbiosis.

Cada vez que el *default* r_1 comete un error, pierde fuerza. Cuando r_2 gana, evitando el error, le evita a r_1 la pérdida. Así, la presencia de r_2, aunque contradice a r_1, en realidad beneficia a r_1. En conjunto, las dos reglas dan al sistema un modelo mucho mejor que el que cada una de éstas proveería por separado.

Al formar modelos internos con la sintaxis propuesta, encontraremos que es más fácil descubrir y probar una regla general que una más particular. Para ver esto, consideremos una regla que tiene 100 detectores (100 bits en la parte de condición). La condición más simple que usa información es aquella que descansa en un solo detector y tiene # en todos los demás. ¿Cuántas condiciones

[20] Llamadas también jerarquías por *default*.
[21] Valor por omisión.

existen que descansan en uno solo de los detectores? Podemos contarlas como sigue.

Escogemos cualquiera de las 100 posiciones como la propiedad que deseamos satisfacer. Es decir, podemos escoger 100 posiciones y hay dos posibilidades por posición. De manera que tenemos 200 condiciones que usan solamente un detector. Todas estas condiciones pueden ser probadas en corto tiempo. En el otro extremo tenemos una condición que usa todos los detectores. Aquí tenemos que escoger una de dos posibilidades para cada una de las 100 posiciones, así que tenemos $2^{100} \approx 10^{30}$ condiciones distintas de este tipo. Este número es mucho mayor que la edad del universo medida en microsegundos. Obviamente, no es posible examinar todas esas condiciones.

Las condiciones generales no solamente son menos, sino que se ensayan más frecuentemente en un medio ambiente típico. Supongamos que todos los mensajes de los detectores son igualmente probables. Entonces, un detector se encontrará en *1* con probabilidad ½, es decir, la mitad de los mensajes tienen un valor, digamos *1*, para un detector dado. Consideremos entonces la condición 1###...#. Ésta se satisfará durante la mitad del tiempo. Está siendo probada muy frecuentemente, de manera que la asignación de crédito rápidamente le asignará una fuerza apropiada a la regla que use esta condición.

Contrastemos la condición 10##...# que es ligeramente más específica. La mitad de los mensajes contendrán un *1* en la primera posición, pero solamente la mitad de ellos tendrá un *0* en la segunda posición. Es decir, solamente ¼ de los mensajes satisfarán 10##...#, de modo que esta condición es probada la mitad de las veces que 1###...# lo es. Es obvio que cada detector que especificamos disminuye la tasa de pruebas en ½.

Claramente, las condiciones útiles generales –*defaults*– son relativamente fáciles de encontrar y establecer. Las reglas más específicas de excepción toman progresivamente más tiempo para encontrar y establecer. Esto sugiere que, bajo una asignación de crédito, el sistema dependerá, al principio, de reglas sumamente generales que son apenas mejores que las acciones aleatorias. A medida que la experiencia se acumula, estos modelos internos serán modificados agregando reglas competidoras de excepción más específicas. Éstas interactuarán simbióticamente con las reglas por *default*. El modelo resultante es denominado una jerarquía por omisión. Éstas se expanden a lo largo del tiempo a partir de valores por omisión muy generales hacia excepciones específicas.

V.4.4.1 Ejemplo de jerarquías por omisión

Para ilustrar el concepto de jerarquías por omisión ya mencionado, consideremos el siguiente problema. Es necesario satisfacer cinco condiciones que denominaremos *v*, *w*, *x*, *y*, *z*. Para que éstas se den es necesario encontrar la combina-

ción correcta de cinco elementos, llamados *A*, *B*, *C*, *D*, *E*, en donde éstos implican a las condiciones de la siguiente manera:

	v	w	x	Y	z
A	**x**		**x**		
B	**x**			**x**	
C		**x**		**x**	
D			**x**		**x**
E	**x**				**x**

Solamente cuando la combinación correcta de *A*, *B*, *C*, *D*, *E* se presente se satisfarán las cinco condiciones *v*, *w*, *x*, *y*, *z*. Por ejemplo, una posible combinación es *ABCDE*. Lo que deseamos es encontrar todas las combinaciones mínimas y completas de *A*, *B*, *C*, *D*, *E*.

Aplicando las reglas del álgebra booleana es simple observar que

$$F(A,B,C,D,E) = (A + B + E\ )C(A + D)(B + C)(D + E)$$
$$= ACD + ACE + BCD + CDE$$

De la relación anterior, y aprovechando la equivalencia $A \supset B \equiv \overline{A} + B$ podemos obtener

$$\alpha = [A \supset \{C \supset \{D \supset \perp\}\}]$$
$$\beta = [A \supset \{C \supset \{E \supset \perp\}\}]$$
$$\gamma = [B \supset \{C \supset \{D \supset \perp\}\}]$$
$$\delta = [C \supset \{D \supset \{E \supset \perp\}\}]$$

de donde:

$$F(A,B,C,D,E) = \alpha \supset \{\ \beta \supset \{\ \gamma \supset \{\ \delta \supset \perp\}\}\}$$

Para efectos de un clasificador, podemos plantear la siguiente representación:

R1 = D $\supset \perp$	R5 = C $\supset$ R4	R9 = B $\supset$ R8
R2 = C $\supset$ R1	R6 = A $\supset$ R5	RA = E $\supset \perp$
R3 = A $\supset$ R2	R7 = D $\supset \perp$	RB = D $\supset$ RA
R4 = E $\supset \perp$	R8 = C $\supset$ R7	RC = C $\supset$ RB

Si asignamos un índice a cada una de las reglas,[22] podemos establecer una tabla como la siguiente.

[22] Eliminamos las reglas R7 y RA ya que R1 = R7 y R4 = RA.

Condición	Indice	Código	Condición	Indice	Código
A	0	0000	R4	8	1000
B	1	0001	R5	9	1001
C	2	0010	R6	10	1010
D	3	0011	R8	10	1011
E	4	0100	R9	12	1100
R1	5	0101	RB	13	1101
R2	6	0110	RC	14	1110
R3	7	0111			

Entonces, un posible conjunto de clasificadores es el siguiente:

Regla	Clasificador
R1	00110:11111
R2	00100:01010
R3	00000:01100
R4	01000:11111
R5	00100:10000
R6	00000:10110
R8	00100:01010
R9	00010:10110
RB	00110:10000
RC	00100:11010
RD	#####:00000

En los clasificadores anteriores, cada condición tiene un bit en la extrema derecha que actúa como una etiqueta. Si es *0* la condición se identifica como un mensaje proveniente de otro clasificador; si es *1* proviene del medio ambiente. De manera análoga, todos los clasificadores que envían un mensaje tienen un *0* en la posición menos significativa si su mensaje activará a otra regla; un *1* si el mensaje actúa en los efectores que afectan al medio ambiente.

La regla RD es general y se aplica cuando ninguna de las anteriores lo hace. Lo que buscamos al establecer un sistema de asignación de crédito con el algoritmo de cascada es, precisamente, que se establezca una jerarquía como la anterior.

El conjunto perfecto de reglas está dado por la siguiente expresión booleana.

$$F(A,B,C,D,E) = \overline{A}BCD\overline{E} + ABCD\overline{E} + A\overline{B}CD\overline{E} + \overline{A}\overline{B}CDE + \overline{A}BCDE + ABCDE + A\overline{B}CDE + ABC\overline{D}E + A\overline{B}C\overline{D}E$$

Esta expresión puede ser escrita en forma canónica, término a término. Por ejemplo, los dos primeros términos son:

$$\overline{A}BCD\overline{E} = \overline{A} \supset \{B \supset \{C \supset \{D \supset \overline{E}\}\}\}$$
$$ABCD\overline{E} = A \supset \{B \supset \{C \supset \{D \supset \overline{E}\}\}\}$$

Dejamos como ejercicio para el lector, la determinación del total de reglas que constituyen el conjunto perfecto. ¿Es más eficiente aplicar dicho conjunto de reglas o aquellas derivadas de una jerarquía por omisión?

V.5 AJUSTE PARAMÉTRICO DE ORDEN LIBRE

A diferencia de los enfoques mencionados en los puntos anteriores, el ajuste paramétrico de orden libre es no supervisado. En los AGs involucrados, además, se busca la evolución no de la solución misma sino de la forma de la solución. En términos generales, el método consiste de lo siguiente:

a) Suponemos que los datos que caracterizan al sistema bajo estudio se pueden encontrar en forma de tabla. En esta tabla existe una columna por cada uno de los parámetros que caracterizan al sistema.
Para efectos de ilustración supondremos los datos de la tabla T.V.5.A.

En la tabla T.V.5.A se anexa un ejemplo. Nótese que la tabla es relativamente extensa y consta de 603 filas (n = 603) y $p+1$ columnas (parámetros) en donde p es el número de variables independientes. En el ejemplo, p = 5.

TABLA T.V.5.A: *Tabla de valores característicos.*

	v_1	v_2	v_3	v_4	v_5	$f(v_1, v_2, v_3, v_4, v_5)$
1	18.31200	30.96213	34.37500	61.37500	10.18800	10.31200
2	18.50000	30.96213	34.25000	62.37500	10.31200	11.25000
3	19.56200	30.96213	34.50000	62.25000	11.25000	11.56200
4	19.43800	30.96213	34.75000	62.62500	11.56200	11.00000
5	18.87500	30.96213	35.50000	62.62500	11.00000	10.62500
6	18.43800	30.96213	35.50000	62.25000	10.62500	10.25000
7	18.50000	30.96213	36.50000	64.37500	10.25000	10.00000
8	18.25000	30.96213	36.62500	64.87500	10.00000	9.62500
9	18.12500	30.96213	36.25000	65.12500	9.62500	10.00000
10	19.25000	30.96213	35.50000	62.87500	10.00000	9.43800
11	19.00000	30.96213	36.25000	63.25000	9.43800	9.37500
601	57.00000	64.87500	69.12500	86.56126	25.00000	26.12500
602	56.87500	64.25000	68.87500	86.56126	26.12500	26.37500
603	57.50000	64.12500	69.00000	86.56126	26.37500	18.34453

b) Deseamos expresar una de las variables en función de las otras *p*.

Por convención consideramos que la *p+1*-ésima variable es la variable dependiente y las otras *p* variables son independientes.

c) Elegimos, *a priori*, la forma del aproximante.

Para efectos del ejemplo supondremos una forma polinomial completa, del tipo

$$f(v_1,\ldots,v_p)=\sum_{i_1=0}^{g_1}\cdots\sum_{i_p=0}^{g_p}C_{i_1\ldots i_p}v_1^{i_1}\cdots v_p^{i_p} \qquad \text{(V.5.A)}$$

en donde g_i denota el máximo grado que puede tomar la *i*-ésima variable independiente.

d) Escogemos, *a priori*, una métrica que nos permita definir la distancia entre el aproximante y los valores de la tabla.

V.5.1 Fundamentos matemáticos[23]

Típicamente, en los problemas de interés los sistemas están sobredeterminados. Es decir, existen más datos que parámetros. Por ello, los procesos de aproximación arrojan ajustes parciales y es necesario medir qué tan lejos estamos de la curva[24] que representa la relación entre las variables independientes y la variable dependiente. En este caso, la norma elegida es la norma L_∞ o minimax. Es decir, deseamos encontrar los coeficientes $C_{0\ldots0},C_{0\ldots1},\ldots,C_{g_1\ldots g_p}$ tales que se minimice

$$\varepsilon_i=(f_{1\ldots p})_i-\left(\sum_{i_1=0}^{g_1}\cdots\sum_{i_p=0}^{g_p}C_{i_1\ldots i_p}v_1^{i_1}\cdots v_p^{i_p}\right)_i \qquad i=1,\ldots,m \qquad \text{(V.5.B)}$$

V.5.1.1 Solución al conjunto M

Elegimos arbitrariamente *m* vectores de la tabla, en donde

---

[23] Ver [16].

[24] Hipercurva *p*-dimensional, en este caso.

$$m = (g_1 + 1)x(g_2 + 1)x...x(g_p + 1) + 1$$

$$m = 1 + \prod_{i=1}^{p} (g_i + 1) \qquad \text{(V.5.C)}$$

De (V.5.B) es fácil escribir

$$(\sum C\vec{v})_i + \varepsilon_i = f_i \qquad \text{(V.5.D)}$$

donde

$$f = f_{1...p}$$
$$C = C_{i1...ip}$$
$$\vec{v} = v_1^{i1} \ldots v_p^{ip}$$

y

$$(\sum C\vec{v})_i = (\sum_{i_1=0}^{g_1} \ldots \sum_{i_p=0}^{g_p} C_{i1...ip} v_1^{i_1} \ldots v_p^{i_p})_i$$

Sean $\varepsilon_\theta = \max\{|\varepsilon_1|,...,|\varepsilon_m|\}$ y $\varepsilon_i = \eta_i \varepsilon_\theta$. Claramente, $\eta_i \leq 1$. Es ε_θ lo que queremos minimizar. De (V.5.D) podemos poner

$$(\sum C\vec{v})i + \eta_i \varepsilon_\theta = f_i \qquad \text{(V.5.E)}$$

Al escribir el sistema de (V.5.E) en forma matricial, tenemos

$$\begin{bmatrix} (v_1^0 \ldots v_p^0)_1 & \ldots & (v_1^{g_1} \ldots v_p^{g_p})_1 & \eta_1 \\ . & \ldots & . & . \\ (v_1^0 \ldots v_p^0)_m & \ldots & (v_1^{g_1} \ldots v_p^{g_p})_m & \eta_m \end{bmatrix} \begin{bmatrix} C_{0...0} \\ \ldots \\ C_{g_1 .. g_p} \\ \varepsilon_\theta \end{bmatrix} = \begin{bmatrix} f_1 \\ \ldots \\ f_m \end{bmatrix} \qquad \text{(V.5.F)}$$

que podemos escribir como

$$\mathbf{A} \cdot \mathbf{C} = \mathbf{f}.$$

De la regla de Cramer, el valor de ε_θ es

$$\varepsilon_\theta = \frac{\begin{vmatrix} (v_1^0 \ldots v_p^0)_1 & \cdots & (v_1^{g_1} \ldots v_p^{g_p})_1 & f_1 \\ . & \ldots & . & . \\ (v_1^0 \ldots v_p^0)_m & \cdots & (v_1^{g_1} \ldots v_p^{g_p})_m & f_m \end{vmatrix}}{\begin{vmatrix} (v_1^0 \ldots v_p^0)_1 & \cdots & (v_1^{g_1} \ldots v_p^{g_p})_1 & \eta_1 \\ . & \ldots & . & . \\ (v_1^0 \ldots v_p^0)_m & \cdots & (v_1^{g_1} \ldots v_p^{g_p})_m & \eta_m \end{vmatrix}} \qquad \text{(V.5.G)}$$

Si denotamos el numerador de (V.5.G) con Δ_f y los cofactores de la última columna del denominador por Δ_i podemos escribir la ecuación (V.5.G) como

$$\varepsilon_\theta = \frac{\Delta_f}{+\eta_1 \Delta_1 + \ldots + \eta_m \Delta_m}$$

Sea $\sigma = sgn(x)$ el signo de *x*. Puesto que deseamos minimizar el valor absoluto de ε_θ debemos escoger los valores que maximicen el valor absoluto del denominador de (V.5.G-H). Esto se logra: *a)* escogiendo el valor máximo absoluto de las η_i's y b) Haciendo $\sigma_i = sgn(\Delta_i)$ o $\sigma_i \neq sgn(\Delta_i)$. Sabemos que $\max\{\eta_1,\ldots,\eta_m\} = 1$. Por lo tanto, el ajuste minimax para el conjunto implica que las ε_i's tienen el mismo valor absoluto y que los signos de tales ε_i's son todos iguales (o todos diferentes) a los signos de las Δ_i's. Es decir, ε_θ se minimiza si

$$\varepsilon_\theta = \frac{\Delta_f}{+\sigma_1 \Delta_1 + \ldots + \sigma_m \Delta_m} \qquad \text{(V.5.H)}$$

En los casos en los que hubiere una sola variable independiente ($p = 1$) las condiciones señaladas se logran trivialmente ordenando los vectores en orden estrictamente ascendente por el valor numérico de la variable independiente. Lo anterior se sigue de que del determinante del denominador de (V.5.G), para ese caso, es un determinante de Vandermonde [9]. Sin embargo, es imposible lograr un ordenamiento de ese tipo si $p > 1$. Por tanto, la manera más sencilla de calcular las σ_i's se sigue del siguiente:

TEOREMA. Si los elementos de una columna de un determinante se multiplican por el cofactor de una columna diferente y se suman, el resultado es cero [13].

Si denotamos los elementos en el determinante del denominador en (V.5.G) como $\delta_{11},\ldots,\delta_{1m};\delta_{21},\ldots,\delta_{2m};\bullet\bullet\bullet;\delta_m 1,\ldots,\delta_{mm}$ podemos poner

$$\Delta_1\begin{bmatrix}\delta_{11}\\ \delta_{12}\\ \bullet\\ \bullet\\ \bullet\\ \delta_{1,m-1}\end{bmatrix}+\Delta_2\begin{bmatrix}\delta_{21}\\ \delta_{22}\\ \bullet\\ \bullet\\ \bullet\\ \delta_{2,m-1}\end{bmatrix}+\bullet\bullet\bullet+\Delta_m\begin{bmatrix}\delta_{m1}\\ \delta_{m2}\\ \bullet\\ \bullet\\ \bullet\\ \delta_{m,m-1}\end{bmatrix}=0$$

o bien

$$\begin{gathered}\Delta_1\delta_{11}+\Delta_2\delta_{21}+\ldots+\Delta_m\delta_{m1}=0\\ \Delta_1\delta_{12}+\Delta_2\delta_{22}+\ldots+\Delta_m\delta_{m2}=0\\ \bullet\bullet\\ \bullet\bullet\\ \bullet\bullet\\ \Delta_1\delta_{1,m-1}+\Delta_2\delta_{2,m-1}+\ldots+\Delta_m\delta_{m,m-1}=0\end{gathered}\qquad\text{(V.5.I)}$$

Este es un sistema con *m* incógnitas y solamente *m-1* condiciones. Podemos elegir, sin embargo, el valor de cualquiera de las Δ_i's (digamos Δ_m) arbitrariamente y resolver el sistema (V.5.I) para las *m-1* Δ's restantes. Para que esto sea posible debemos suponer que todas las Δ_i's son no singulares. Cuando un sistema manifiesta esta independencia lineal, se dice que satisface la condición de Haar.

Si hacemos que $\Delta_m = -1 = Z_m$ el sistema de (V.5.1.I) se convierte en

$$\begin{bmatrix}(v_1^0\ldots v_p^0)_1 & \cdots & (v_1^0\ldots v_p^0)_{m-1}\\ \bullet\bullet\bullet & \bullet\bullet\bullet & \bullet\bullet\bullet\\ (v_1^{g_1}\ldots v_p^{g_p})_1 & \cdots & (v_1^{g_1}\ldots v_p^{g_p})_{m-1}\end{bmatrix}\begin{bmatrix}Z_1\\ \bullet\bullet\bullet\\ Z_{m-1}\end{bmatrix}=\begin{bmatrix}(v_1^0\ldots v_p^0)_m\\ \bullet\bullet\bullet\\ (v_1^{g_1}\ldots v_p^{g_p})_m\end{bmatrix}\qquad\text{(V.5.J)}$$

en donde $Z_i = k\Delta_i$. Es decir, no obtenemos el valor de los cofactores, sino el valor de los mismos multiplicado por una constante *k*, la cual puede tomar cualquier valor positivo o negativo distinto de cero. Las σ_i's se obtienen, entonces, de

$$\sigma_i = \begin{cases} \text{sgn}(Z_i) & i \neq m \\ \text{sgn}(\Delta_m) & i = m \end{cases}$$

Sin embargo, sin importar qué valor se asigna a Δ_m (y del cual *k* depende), las σ_i's permanecerán constantes dependiendo de σ_m. Que σ_m puede ser arbitraria se sigue del hecho de que solamente es relevante el signo de Z_i, y no su magnitud y de que, por otra parte, el denominador de (V.5.H) puede maximizarse escogiendo ya sea $\sigma_i = sgn(\Delta_i)\forall i$ o $\sigma_i \neq sgn(\Delta_i)\forall i$.

Podemos entonces escribir la matriz **A** de (V.5.F) como

$$\mathbf{A} = \begin{bmatrix} (v_1^0 \ldots v_p^0)_1 & \cdots & (v_1^{g_1} \ldots v_p^{g_p})_1 & \sigma_1 \\ . & \cdots & . & . \\ (v_1^0 \ldots v_p^0)_m & \cdots & (v_1^{g_1} \ldots v_p^{g_p})_m & \sigma_m \end{bmatrix} \qquad \text{(V.5.K)}$$

y los coeficientes resultantes de $\mathbf{C} = \mathbf{f}\,\mathbf{A}^{-1}$ son los coeficientes minimax que buscamos para un conjunto de *m* vectores.

V.5.1.2 Un ejemplo numérico

Los conceptos anteriores pueden ilustrarse con el siguiente ejemplo. Sean

$$A^1 = (-2, +2, +2)$$
$$A^2 = (+2, +3, +3)$$
$$A^3 = (+3, +1, -1)$$

los vectores de datos que se desean caracterizar. Estos vectores dan origen al siguiente sistema de ecuaciones:

$$-2X_1 + 2X_2 = +2$$
$$+2X_1 + 3X_2 = +3$$
$$+3X_1 + X_2 = -1$$

En la figura F.V.5.A se ilustran las posiciones de los tres puntos correspondientes a los miembros izquierdos de las ecuaciones anteriores.

Este sistema es sobredeterminado y planteamos el siguiente sistema de aproximación.

$$+2 - (-2X_1 + 2X_2) = \varepsilon_1$$
$$+3 - (+2X_1 + 3X_2) = \varepsilon_2$$
$$-1 - (+3X_1 + X_2) = \varepsilon_3$$

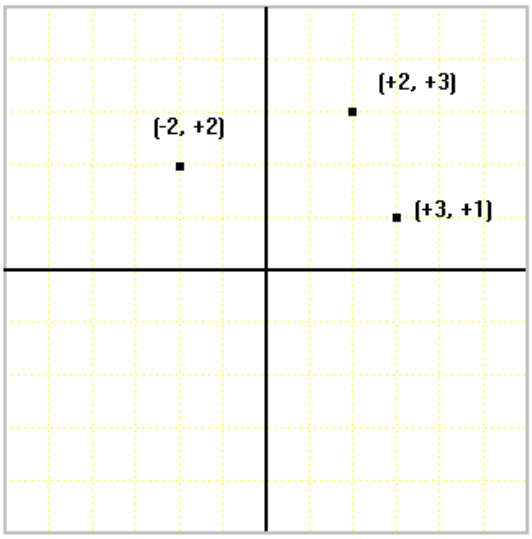


FIGURA F.V.5.A: *Puntos del vector original.*

de donde

$$\begin{aligned} +2X_1 - 2X_2 - \sigma_1\varepsilon &= -2 \\ -2X_1 - 3X_2 - \sigma_2\varepsilon &= -3 \\ -3X_1 - X_2 - \sigma_3\varepsilon &= +1 \end{aligned} \qquad \text{(V.5.L)}$$

Los valores de las σ_i's podemos obtenerlos de:

$$\begin{aligned} -2\Delta_1 + 2\Delta_2 + 3\Delta_3 &= 0 \\ +2\Delta_1 + 3\Delta_2 + \Delta_3 &= 0 \end{aligned}$$

Hacemos $Z_3 = -1$

de donde

$$\begin{aligned} -2Z_1 + 2Z_2 &= +3 \\ +2Z_1 + 3Z_2 &= +1 \end{aligned}$$

cuya solución es

$$Z_1 = -\frac{7}{10}$$

$$Z_2 = +\frac{4}{5}$$

De las Z_is se obtienen las siguientes σ_i's: $\sigma_1 = -1; \sigma_2 = +1; \sigma_3 = -1$. De (V.5.L), entonces:

$$+2X_1 - 2X_2 + \varepsilon = -2$$

$$+2X_1 - 3X_2 - \varepsilon = -3$$
$$-3X_1 - X_2 + \varepsilon = +1$$

En la figura F.V.5.B se ilustran los puntos con los signos correspondientes. Nótese que ahora el origen está en el envolvente convexo del conjunto. Además, $\varepsilon_\theta = +\frac{4}{5}$.

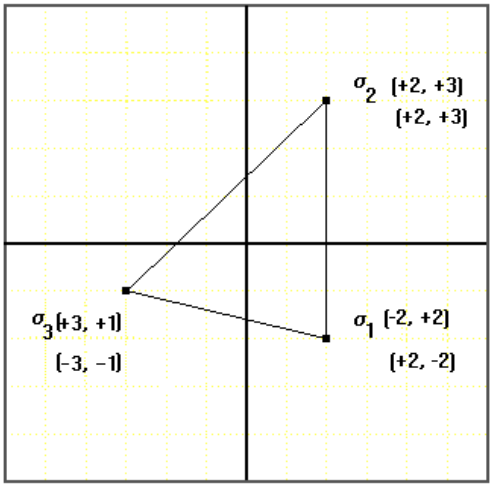


FIGURA F.V.5.B: *Puntos del vector minimax.*

V.5.1.3 Solución al conjunto N

Lo que nosotros buscamos es el conjunto de coeficientes **C** que minimicen ε_θ para los *n* elementos del conjunto original. En la discusión precedente, como ya se anotó, supusimos que los *m* vectores elegidos son arbitrarios. Para encontrar la ε_θ global, es decir, aquella que resuelva el ajuste minimax para los *n* vectores originales, planteamos un algoritmo iterativo de ascenso, análogo al algoritmo de Remes [14]. El algoritmo es el siguiente.

V.5.1.4 Algoritmo de ascenso

1. Hacer $i \leftarrow 1$.
2. Elegir un conjunto de vectores arbitrario M_i de tamaño *m*.
   A este conjunto lo llamamos el conjunto interno; a los vectores no incluidos en este conjunto los agruparemos en otro conjunto E_i que llamamos el *conjunto externo* y que es, claramente, de tamaño *n - m*.

3. Encontrar la matriz $\mathbf{A}_i$ de acuerdo con (V.5.J-K).
4. Encontrar los coeficientes minimax $\mathbf{C}_i$ y el máximo error de ajuste $[\varepsilon_\theta]_i$ para el conjunto interno.
5. Calcular el máximo error de ajuste $[\varepsilon_\phi]_i$ para el conjunto externo.
   En donde

$$[\varepsilon_\phi]_i = \max\{|(f_{1\ldots p})_j - (\sum_{i_1=0}^{g_1}\ldots\sum_{i_p=0}^{g_p} C_{i_1\ldots i_p} v_1^{i_1}\ldots v_p^{i_p})_j|\}$$

$$\vec{v}_j \in E_i$$

6. Si $[\varepsilon_\theta]_i \geq [\varepsilon_\phi]_i$, *fin del algoritmo*.
7. {Si $[\varepsilon_\theta]_i < [\varepsilon_\phi]_i$}
   Intercambiar un vector en M_i por un vector en E_i tal que

   $|[\varepsilon_\theta]_{i+1}| > |[\varepsilon_\theta]_i|$.

8. Hacer $i \leftarrow i+1$.
9. Ir al paso *3*.

Fin del algoritmo de ascenso.

Que el algoritmo converge es claro ya que, por definición,

$$i < \tau \Rightarrow |[\varepsilon_\theta]_i| < |[\varepsilon_\phi]_i|$$

en donde τ denota el índice del conjunto objetivo (M_τ). Esta noción puede formalizarse en el siguiente teorema.

TEOREMA DEL INTERCAMBIO

Sea $\{A^1,\ldots, A^{m+1}\}$ un conjunto de vectores[25] en el espacio m que satisfacen la condición de Haar. Si **0** está en el *envolvente convexo* de $\{A^1,\ldots, A^m\}$, entonces hay un índice $j \leq m$ tal que esta condición se mantiene cuando A^j es reemplazado por A^{m+1} (véase [14], p. 96).

En el algoritmo de ascenso el paso crítico es el 7. Determinar si $|[\varepsilon_\theta]_{i+1}| > |[\varepsilon_\theta]_i|$ (la condición de intercambio o CI) puede lograrse, directamente, reemplazando una por una las filas de la matriz **A** y calculando cada una de las Δs y sus signos de manera que $|[\varepsilon_\theta]_{i+1}| > |[\varepsilon_\theta]_i|$. Tomando en cuenta

[25] Denotamos la *i*-ésima fila de una matriz (digamos **A**) con A^i. De la misma forma, denotamos la *j*-ésima columna de **A** con A_j.

que la solución de un determinante tiene un costo de $O(m^3)$ *flops*,[26] el costo promedio para determinar la *CI* es $\frac{m}{2}O(m^3) = O(m^4)$. Es decir, hay que ejecutar $O(m^4)$ *flops* por iteración. Si suponemos que el número de iteraciones crece de manera cuadrática con *n*, entonces el costo promedio del algoritmo de ascenso (C_{asc}) puede ser aproximado por

$$C_{asc} = O(n^2 \bullet m^4) > O(m^6)$$
$$C_{asc} = km^6 \qquad \text{(V.5.M)}$$

Por ejemplo, si $m = 60$,

$$C_{asc} \approx 60^6 \approx 6^6 \text{ X } 10^6$$
$$C_{asc} \approx 1.7 \text{ x } 10^{12}$$

Para lograr un sistema más eficiente, es posible usar técnicas numéricas y heurísticas que nos permiten disminuir C_{asc} hasta $O(k\, m^2)$ [16].

V.5.2 Solución de sistemas singulares

El método delineado en la sección V.5.1 se basa en la obtención de un mapa de cada uno de los vectores del problema original en R^p a R^m (en nuestro ejemplo cada vector consta de 6 elementos, es decir, $p = 5$) y la posterior obtención de los σ_i para conformar **A**. Del teorema del intercambio se implica que cada uno de los conjuntos de vectores de **A** cumpla con la condición de Haar. ¿Cuál es la posibilidad de que alguno de los conjuntos no cumpla con esta condición? Y si este fuera el caso, ¿existe alguna forma de encontrar la solución a los llamados *sistemas singulares*?

Una vez conformada la matriz **A** cuyos vectores están en R^m, ésta no satisface la condición de Haar si

$$A^i = K\, A^j ; (i, j) \le p; i \ne j; K \equiv constante$$

ni cuando

$$A_j = K\, A_j ; (i, j) \le p; i \ne j; K \equiv constante$$

Si podemos garantizar que ninguna de tales filas o columnas sean linealmente dependientes la condición de Haar puede mantenerse. Para lograr esto inducimos perturbaciones aleatorias en los vectores $v_i = [(v_1)_i, \ldots, (v_p)_i, (v_{p+1})_i]$ del conjunto original *N* haciendo

[26] Operaciones de punto flotante.

$$(V_1)_i = (v_1)_i(1+\delta_i)$$
$$(V_2)_i = (v_2)_i(1+\delta_{n+i})$$
$$.....$$
$$(V_{p+1})_i = (v_{p+1}(1+\delta_{np+i})$$

$$\delta_i = \delta_H \; x \; \rho_i$$
en donde :
$$\rho_i = aleat() \quad [i = 1,...,n(p+1)]$$
y
$$0 < \rho_i < 1$$
$$\delta_H << 1$$

En este esquema δ_H es constante, por tanto, cada vector $\boldsymbol{v}_i$ en el conjunto original N es reemplazado por otro vector $\boldsymbol{V}_i = [\,(V_1)_i\,,\,(V_2)_i\,,\,...,\,(V_{p+1})_i\,]$ el cual exhibe un desplazamiento porcentual aleatorio relativo a $\boldsymbol{v}_i$ que está acotado por δ_H. El conjunto resultante N^* satisface la condición de Haar para una δ_H apropiada aún si N no lo hace.

Una interpretación geométrica del método descrito consiste en visualizar cada uno de los puntos en R^m de manera que ningún subconjunto de ellos sea colineal en ninguna de las m dimensiones. Para evitar lo anterior estamos desplazando ligeramente cada punto de manera aleatoria. El hacer el desplazamiento aleatoriamente evita que se mantenga una relación lineal entre los vectores.

En la figura F.V.5.C se ilustra el caso de dependencia lineal para $p=2$. Los tres puntos que se encuentran unidos por la línea recta, se supone, establecen que al menos dos filas y/o columnas de **A** son proporcionales.

En la figura F.V.5.D se ilustra el desplazamiento aleatorio de cada uno de los tres puntos colineales de la figura anterior. Por definición los desplazamientos no conservan la relación de proporcionalidad que da origen a la dependencia lineal y al hecho de que alguno de los menores de (V.5.H) sea igual a cero, con el consecuente resultado de que no sería posible calcular el valor de ε_θ a partir de la ecuación (V.5.G).

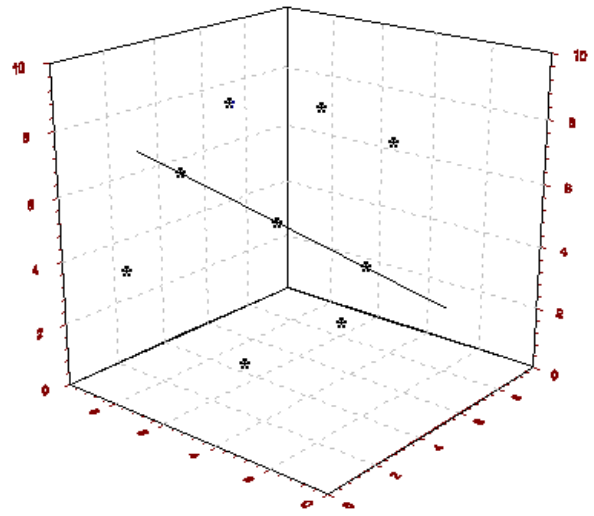


FIGURA F.V.5.C: *Dependencia lineal.*

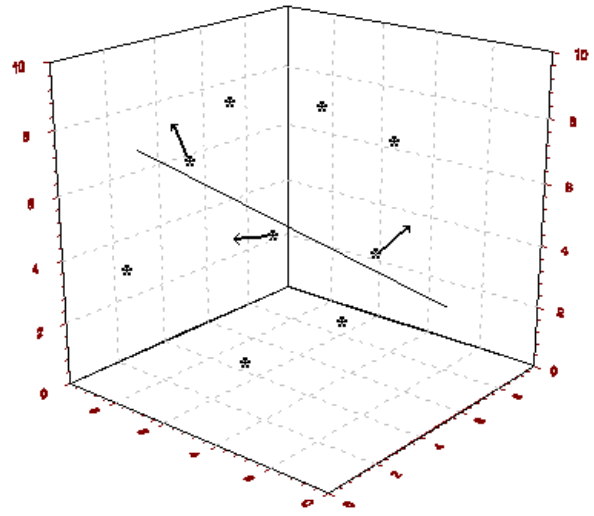


FIGURA F.V.5.D: *Desplazamiento aleatorio.*

La sustitución de ***v*** por ***V*** pretende resolver el problema de la dependencia lineal, como ya se mencionó. Sin embargo, este es solamente un heurístico. En algunos casos no se logra la independencia deseada de manera directa. Por esta razón, en la práctica se emplea el siguiente algoritmo.

ALGORITMO DE CONDICIONAMIENTO

desde i=1 **hasta** η
  mientras cierto
    lee *N* datos originales
    de *N* datos originales obten N^*
    calcula **C**
    si sistema es linealmente dependiente
      $\delta_H = \delta_H \times \Phi$
      si $\delta_H > \mu_\pi$
        $\rho = aleat() \{0 < \rho < 1\}$
        $\Phi = \Phi x(1 + \rho)$
        $\delta_H = (\delta_H)_0 \bullet \Phi$
        siguemientras
      finsi
    {si no}
      saldesde
    finsi
  finmientras
findesde

El algoritmo anterior recibe cuatro parámetros de entrada:

a) El valor del factor de perturbación inicial $(\delta_H)_0$, en donde $\delta_H << 1$.
b) La máxima perturbación aceptable (μ_π).
c) Un factor de ajuste (Φ).
d) El máximo número de ciclos de intento de estabilización (η).

Empieza perturbando el conjunto de vectores y verifica que el sistema resultante haya cumplido con la condición de Haar. Si éste no es el caso, aumenta el factor de perturbación por una cantidad constante y obtiene un nuevo conjunto N^*. Típicamente, $(\delta_H)_0 = 10^{-6}$ y $\Phi = 2$. Eso significa que, en iteraciones sucesivas, $\delta_H = 2x10^{-6}, 4x10^{-6}, \ldots$ Este proceso continúa mientras no se logre la estabilidad del sistema. Sin embargo, si el sistema excede un valor máximo de perturbación aceptable μ_π (por supuesto no es deseable que los vectores originales se perturben más allá de cierto punto) el algoritmo intenta un nuevo factor de ajuste cuyo valor es aleatorio (lo cual cierra un ciclo) y el proceso se repite. Este ciclo de repeticiones está acotado por η.

El algoritmo anterior es un heurístico que no garantiza que se logre la estabilidad deseada. Esto depende, básicamente, de μ_π. Sin embargo, en prácticamente todos los casos en que ha sido probado, ha logrado sistemas estables. Esto es

muy interesante ya que, por la forma del aproximante definida en (V.5.A) es claro que la dependencia lineal es altamente probable.

Obviamente, cuando ajustamos a estos nuevos datos solamente estamos aproximándonos a la solución requerida. Ahora estableceremos una cota en la diferencia entre la aproximación minimax a N^* en V_τ y en v_τ.[27] Aquí v_τ y V_τ denotan cualquier vector en los conjuntos objetivo M_τ y M^*_τ para N y N^*, respectivamente. Sea

$$F^*_\tau = \sum_{i_1=0}^{g_1} \ldots \sum_{i_p=0}^{g_p} C_{i_1 \ldots i_p} (V_1^{i_1})_\tau \ldots (V_p^{i_p})_\tau$$

el valor del polinomio minimax obtenido de N y evaluado en cualquier punto del conjunto M^*_τ; sea

$$F_\tau = \sum_{i_1=0}^{g_1} \ldots \sum_{i_p=0}^{g_p} C_{i_1 \ldots i_p} (v_1^{i_1})_\tau \ldots (v_p^{i_p})_\tau$$

el valor del polinomio minimax obtenido de N y evaluado en cualquier punto de M_τ; y $TDG(x) \equiv$ *términos de grado x en* F_τ; $TDO(x) \equiv$ *términos de orden x en* F_τ. Entonces,

$$F^*_\tau = (1+\delta_H)^0 \, xTDG(0) +$$
$$(1+\delta_H)^1 \, xTDG(1) +$$
$$\bullet\bullet\bullet$$
$$(1+\delta_H)^{g_1+\ldots+g_p} \, xTDG(g_1+\ldots+g_p)$$

en donde, por simplicidad, hemos tomado el máximo valor de δ_i, es decir, δ_H.

Expandiendo el valor de los binomios, tenemos

$$F^*_\tau = TDG(0) + \ldots + TDG(g_1+\ldots+g_p)$$
$$+ \delta_H \, [1xTDG(1) + \ldots + (g_1+\ldots+g_p) xTDG(g_1+\ldots+g_p)]$$
$$+ TDO(\delta_H^2) + \ldots + TDO(\delta_H^{g_1+\ldots+g_p})$$

Ya que $\delta_H << 1$ podemos escribir

$$F^*_\tau \approx TDG(0) + \ldots + TDG(g_1+\ldots+g_p)$$
$$+ \delta_H \, [1xTDG(1) + \ldots + (g_1+\ldots+g_p) xTDG(g_1+\ldots+g_p)]$$

---

[27] El subíndice τ denota el conjunto objetivo. M_τ denota, pues, el conjunto interno objetivo, E_τ el conjunto externo objetivo, etcétera.

$$F_\tau^* - F_\tau = \delta_H\,[1xTDG(1) + \ldots + (g_1 + \ldots + g_p)xTDG(g_1 + \ldots + g_p)]$$

y, sucesivamente,

$$\delta_H \sum_{i=1}^{g_1+\ldots+g_p} ixTDG(i) = \delta_H\{[F_\tau - TDG(0)] + [F_\tau - TDG(0) - TDG(1)] + \ldots +$$
$$[F_\tau - TDG(0) - TDG(1) - \ldots - TDG(g_1 + \ldots + g_p - 1)]\}$$
$$= \delta_H \sum_{i=1}^{g_1+\ldots+g_p} \{F_\tau - \sum_{j=0}^{i-1} TDG(j)\}$$

Tomando valores absolutos,

$$|F_\tau^* - F_\tau| \approx \left| \delta_H \sum_{i=1}^{g_1+\ldots+g_p} [F_\tau - \sum_{j=0}^{i-1} TDG(j)] \right| \qquad \text{(V.5.N)}$$

Puesto que $\delta_H > 0$ y $|\sum_i a_i| \leq \sum_i |a_i|$ entonces

$$|F_\tau^* - F_\tau| \leq \delta_H \sum_{i=1}^{g_1+\ldots+g_p} \left| F_\tau - \sum_{j=0}^{i-1} TDG(j) \right| < \delta_H\, x(g_1 + \ldots + g_p)\,|F_\tau|$$

De donde concluimos

$$\left| \frac{F_\tau^* - F_\tau}{F_\tau} \right| < [\sum_i g_i]\,\delta_H \qquad \text{(V.5.O)}$$

Por otra parte, podemos escribir

$$(i) \quad f_\tau - F_\tau = \varepsilon_\theta$$
$$(ii) \quad f_\tau + f_\tau \delta_H - F_\tau^* = \varepsilon_\theta^*$$

Restando *(ii)* de *(i)*

$$F_\tau^* - F_\tau - f_\tau \delta_H = \varepsilon_\theta - \varepsilon_\theta^*$$

Sabemos, de (V.5.N) que

$$F_\tau^* - F_\tau - f_\tau \delta_H = \delta_H \left[\sum_{i=1}^{g_1+\ldots+g_p} \{F_\tau - \sum_{j=0}^{i-1} TDG(j)\} - f_\tau \right]$$

por lo tanto

$$|\varepsilon_\theta - \varepsilon_\theta^*| \leq \delta_H \{ \sum_{i=1}^{g_1+\ldots+g_p} |F_\tau - \sum_{j=0}^{i-1} TDG(j)| + |- f_\tau|\}$$

$$|\varepsilon_\theta - \varepsilon_\theta^*| < \delta_H \{|(g_1+\ldots+g_p)|F_\tau| + |f_\tau|\}$$

y como $|f_\tau| < |F_\tau| x2$; $\varepsilon_\theta > 0$ y $\varepsilon_\theta^* > 0$ entonces

$$|\varepsilon_\theta^* - \varepsilon_\theta| < (2 + [\sum_i g_i])\delta_H |F_\tau| \qquad \text{(V.5.P)}$$

Esencialmente, pues, las ecuaciones (V.5.O) y (V.5.P) nos permiten concluir que el proceso de condicionamiento lleva a una diferencia relativa entre la función evaluada en N^* y la función evaluada en N del orden de δ_H. Por otra parte, la diferencia de los errores minimax (ε_θ) en N^* y N es, también, del orden de δ_H. Puesto que el valor de δ_H es, por definición, muy chico, el aproximante que se obtiene del conjunto perturbado aleatoriamente es muy cercano al aproximante original obtenido de N.

Las desigualdades de (V.2.O) y (V.5.P) son bastante sueltas. Recordemos que aproximamos las δ_i con el máximo valor δ_H; el valor esperado es ½ δ_H. No hay que olvidar, además, que por la propia naturaleza del algoritmo, el error minimax en M_τ es siempre mayor que el error minimax en E_τ.

V.5.3 Solución de sistemas completos

Usando el método señalado en V.5.2, es factible caracterizar un sistema de datos arbitrariamente obtenidos. Lo anterior es fundamentalmente importante porque, típicamente, se imponen ciertas demandas a la forma de los datos para lograr una expresión funcional como la de (V.5.A). Cuando logramos encontrar dicha expresión, lo que estamos haciendo, de hecho, es sintetizar de manera muy compacta nuestro conocimiento del sistema. En ese sentido es que el método "aprende" de la experiencia pasada (los datos históricos). Como puede observarse, el método de aproximación atiende a una selección *a priori* de la forma del aproximante y la norma de aproximación. Aquí queremos consignar que el método descrito se puede aplicar sin modificaciones a otro tipo de aproximantes. Una alternativa destacable son los aproximantes de la forma

$$f(v_1,\ldots,v_p) = \sum_{i_1=0}^{g_1} \cdots \sum_{i_p=0}^{g_p} \frac{C_{i_1 \ldots i_p}}{v_1^{i_1} \cdots v_p^{i_p}}$$

y que son, ya que incluyen polos, más ricos que los aproximantes polinomiales.

En la exposición anterior hemos omitido un hecho práctico de fundamental importancia. En un aproximante de la forma de (V.5.A), el número de coeficientes crece exponencialmente con el número de variables independientes y los grados máximos estipulados. Por ejemplo, para 3 variables y $g_1 = g_2 = g_3 = 4$ el número de coeficientes es $5^3 = 125$; para 10 variables y $g_1 = ... = g_{10} = 6$ el número de coeficientes es $7^{10} \approx 7{,}282.475 \times 10^6$. Evidentemente, una solución cuya caracterización implica tal número de datos es inútil. Adicionalmente, la solución de los sistemas de ecuaciones de gran tamaño induce problemas de estabilidad numérica que invalidan los resultados obtenidos cuando los rangos dinámicos en los sistemas (V.5.F) son grandes.

Para evitar el problema mencionado, reemplazamos el sistema de (V.5.A) por el siguiente:

$$f(v_1,...,v_p) = \sum_{i_1=0}^{g_1} ... \sum_{i_p=0}^{g_p} \mu_{i_1..i_p} C_{i_1..i_p} v_1^{i_1} ... v_p^{i_p} \qquad \text{(V.5.Q)}$$

en donde las variables $\mu_{i_1,...i_p}$ solamente pueden tomar los valores 0 ó 1 y su propósito es excluir/incluir el coeficiente del cual son factor. Inducimos un nuevo parámetro en el problema: el número máximo de coeficientes deseados (γ). Es decir, ahora el problema consiste en: *a)* encontrar los γ coeficientes más convenientes para minimizar ε_θ, y *b)* encontrar los valores de los coeficientes elegidos. Buscamos, ahora, dos vectores de solución. El vector $\boldsymbol{\mu}$ y el vector **C**. El vector $\boldsymbol{\mu}$ tiene $1 + \prod_{i=0}^{p} (g_i + 1)$ elementos; el vector **C** tiene γ elementos. Estos son problemas altamente no lineales y NP completos. Ambos pueden ser resueltos mediante un algoritmo genético, como se describe a continuación.

V.5.4 Algoritmo genético de orden

El problema que nos ocupa puede ser resuelto si establecemos un genoma en el cual observemos la siguiente convención:

a) Tiene una longitud *m* igual al número de coeficientes de la expresión (V.5.Q).
b) Cada uno de los bits del genoma corresponde con un coeficiente de (V.5.Q). Los coeficientes se indizan de manera canónica de izquierda a derecha, de manera que los índices corren desde 0 hasta *m*.

Por ejemplo, si $p = 3$ y $g_1 = 1$, $g_2 = 2$ y $g_3 = 2$, entonces:

Potencias asociadas a los coeficientes

000	001	002	010	011	012	020	021	022	100	101	102	110	111	112	120	121	122

Índices de los coeficientes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

c) Existen γ posiciones en las cuales el genoma toma el valor 1; m - γ posiciones en las cuales el genoma toma el valor 0. Al número de 1's del genoma lo denominaremos el *orden* del genoma y será representado por ω.

d) Un 1 en el genoma significa que el coeficiente correspondiente existe (μ = 1); por el contrario, un 0 en el genoma significa que el coeficiente no existe (μ = 0).

Para el caso anterior, el genoma

000	001	002	010	011	012	020	021	022	100	101	102	110	111	112	120	121	122
1	1	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	1

corresponde al aproximante de la forma

$$f(v_1,v_2,v_3) = C_{000} + C_{001}v_3 + C_{020}v_2^2 + C_{022}v_2^2v_3^2 + C_{101}v_1v_3 + C_{111}v_1v_2v_3 + C_{112}v_1v_2v_3^2 + C_{122}v_1v_2^2v_3^2$$

De la misma manera, el genoma

000	001	002	010	011	012	020	021	022	100	101	102	110	111	112	120	121	122
0	1	0	1	0	0	1	0	1	0	0	0	0	1	0	1	0	1

corresponde al aproximante de la forma

$$f(v_1,v_2,v_3) = C_{001}v_3 + C_{010}v_2 + C_{020}v_2^2 + C_{022}v_2^2v_3^2 + C_{111}v_1v_2v_3 + C_{120}v_1v_2^2 + C_{122}v_1v_2^2v_3^2$$

Sea $\Gamma = (g_1,g_2,...,g_p)$ un conjunto de grados máximos de cardinalidad p y $f(v_1,...,v_p)$ el aproximante minimax de (V.5.Q). Como sabemos, la longitud del genoma está dada por $\prod_{i=1}^{p}(g_i+1)$. Dado el índice de los coeficientes del geno-

ma, es posible encontrar las potencias asociadas a los coeficientes de la siguiente relación:

$$\tau_i = \left\lfloor \frac{(N-1) - \sum_{j=i+1}^{p} \tau_j \prod_{k=1}^{j-1} (g_k + 1)}{\prod_{j=0}^{i-1} (g_j + 1)} \right\rfloor \qquad \text{(V.5.R)}$$

en donde $a > b \Rightarrow \sum_{i=a}^{b} x \equiv 0$ y $g_0 \equiv 0$. Los términos de las potencias ($\tau_p, \tau_{p-1}, \ldots, \tau_1$) corresponden a las posiciones *p, p-1, ... ,1* de la potencia. Por ejemplo, los términos $\tau_3 = 0; \tau_2 = 2; \tau_1 = 1$ en el genoma ilustrado corresponden al índice *8*.

De manera análoga, dados los términos $\tau_p, \ldots, \tau_1$ es posible encontrar el índice correspondiente de la expresión

$$N = 1 + \sum_{i=1}^{p} \tau_i \prod_{j=0}^{i-1} (g_j + 1) \qquad \text{(V.5.S)}$$

en donde, como antes, $g_0 \equiv 0$.

Estos genomas son susceptibles de ser tratados genéticamente de manera que la función de ajuste sea el valor del error ε_θ y el proceso es un proceso de minimización de dicho error. En cada generación se derivan dos componentes para cada individuo:

a) El máximo error de ajuste (ε_θ).
b) El vector de coeficientes **C** para el algoritmo de ascenso.

De esta manera, la población evoluciona hacia la combinación de coeficientes que minimice ε_θ bajo la condición de que el número de 1's en el genoma sea igual a γ.

Es importante hacer notar que el algoritmo genético, en este caso, tiene una función objetivo en la cual, además de minimizar el error ε_θ, debe mantener un genoma cuyo número de 1's sea siempre igual a γ ($\omega = \gamma$). Es decir, los operadores genéticos de cruzamiento y mutación están limitados en el sentido de que éstos no dan origen a genomas válidos en todos los casos. Consecuentemente, es preciso adoptar una de las siguientes estrategias:

a) Establecer una medida de castigo para aquellos genomas (resultantes de los operadores genéticos).
b) Modificarlos para que el orden del genoma se mantenga constante.

En este último caso, debe introducirse un algoritmo de reparación (véase la sección 3.2.3) de manera que los genomas que violen la restricción $\omega = \gamma$ sean reemplazados por otros que no la violen.

La primera estrategia es más general pero la segunda, obviamente, es más eficiente. En este contexto, debemos de preferir la segunda sobre la primera por razones de economía.

Como se ve, pues, el ajuste paramétrico de orden libre no busca la solución del problema (el vector **C**) de manera directa. La búsqueda de la forma de la solución (el mejor individuo de orden γ), sin embargo, nos conduce a la solución del problema de fondo, que es encontrar el conjunto de coeficientes que mejor ajusten los datos.

¿Podemos entender al conjunto **C** como una metodología de aprendizaje y, en última instancia, como una técnica de inteligencia artificial? Hasta este punto, lo que el algoritmo nos permite es *caracterizar* los datos históricos. Este hecho es, *per se*, de mucho interés ya que, con la expresión sintética derivada de **C**, somos capaces de explorar áreas desconocidas del espacio de datos. Sin embargo, nuestra principal motivación es la de establecer una metodología que nos permita explorar áreas *fuera* del dominio de los datos originales. En otras palabras, con la metodología descrita estamos en posibilidades de interpolar. ¿Podemos, también, extrapolar?

Evidentemente, una vez obtenido **C**, el polinomio resultante es susceptible de aceptar cualquier argumento, aún aquellos que se encuentran fuera de los límites del conjunto de datos original. Sin embargo, debido a las relaciones altamente no lineales entre los términos del polinomio aproximante, es difícil que los argumentos fuera del dominio de datos arrojen resultados muy significativos. No obstante, existe una estrategia simple que nos permite subsanar la limitación anterior.

Como se ilustra en la tabla T.V.5.A, es posible expresar una de las variables en función de sí misma, desplazada con respecto a las entradas de la tabla. En ésta, los datos correspondientes a $v_1,...,v_5$ constituyen un registro histórico en donde cada fila de la tabla corresponde a un momento del tiempo y $f(v_1,...,v_5)$ corresponde al valor de v_5 al tiempo $t+1$. Es decir,

$$f(v_1,...,v_5) = v_5(t+1) = f(v_1(t),...,v_5(t))$$

En general,

$$f_i(v_1,...,v_\eta) = v_i(t+\Delta t) = f(v_1(t),...,v_\eta(t)) \qquad i = 1,...,\eta \qquad \text{(V.5.T)}$$

en donde Δt determina qué tantos intervalos de tiempo deseamos dejar pasar entre el momento de observación del fenómeno en cuestión y el momento de predicción. De hecho, es posible plantear un conjunto de variantes al respecto. Po-

demos poner la variable dependiente (que llamaremos variable de prospección) en función de otras variables y de sí misma en distintos instantes de tiempo:

$$f_k(v_1,...,v_{\eta+\tau}) = v_k(t+\Delta t) = f(v_1(t),...,v_k(t),v_k(t+1),...,v_\eta(t))$$
$$\tau = 0,...,n$$

en donde τ especifica cuántas instancias de la variable independiente – 1 aparecen en la tabla de datos.

De manera que el algoritmo genético, junto con el algoritmo de intercambio y el método de condicionamiento de datos nos permiten encontrar una expresión matemática de la relación que existe entre un número indeterminado de variables y una variable de prospección. Evidentemente, este proceso puede ser llevado a cabo de forma más genérica encontrando no una variable de prospección sino un conjunto de variables de prospección, como se esbozó en (V.5.T). El número de posibilidades y variantes es prácticamente ilimitado.

Para concluir, deseamos enfatizar que la metodología descrita es posible, básicamente, debido al método de perturbación aleatoria. Sin éste, las dependencias lineales que se presentan con alta probabilidad en ciertas de las variantes descritas, no podrían ser resueltas. Esta estrategia de perturbación puede ser aplicada para otro tipo de aproximantes y otro tipo de métricas.

V.6 EJERCICIOS

COEVOLUCIÓN

En las páginas subsecuentes se hace una brevísima introducción al problema de entrenamiento de una red neuronal [19]. Sugerimos una posible alternativa al algoritmo llamado de *retropropagación* que es usado frecuentemente. Al término de dicha descripción se encuentra el problema en donde se puede aplicar el concepto de aprendizaje coevolutivo.

ANTECEDENTES

Un perceptrón consiste de una neurona con pesos adaptables w_j $j=1,...,n$ y un umbral u, como se muestra en la figura F.E.1.A.

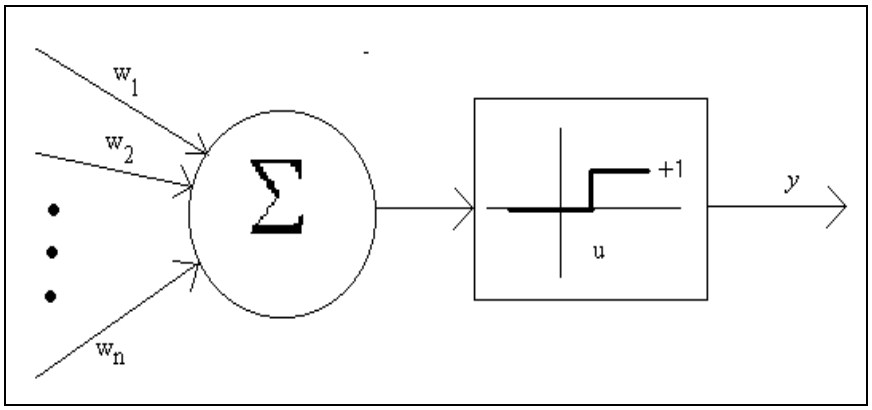


FIGURA F.E.1.A: *Modelo de una neurona (perceptrón).*

Dado un vector de entradas $\boldsymbol{x} = (x_1,...,x_n)$ la entrada neta a la neurona es

$$v = \sum_{j=1}^{n} w_j x_j - u$$

La salida del perceptrón es +1 si *v>0* y *0* en otro caso. Puede demostrarse que el perceptrón, como unidad lógica, es incompleto. Es decir, existen ciertas funciones lógicas básicas que un perceptrón no puede calcular.

Sin embargo, el perceptrón multicapa (PM) que se ilustra en la figura F.E.1.B puede formar fronteras de decisión de complejidad arbitraria y representar cualquier función booleana. En esta figura, el conjunto de entradas está representado por círculos sólidos. Las entradas y, posteriormente, cada neurona está fuertemente conectada con las neuronas de la siguiente capa. En una red neuronal como la que se ilustra, se puede asociar un elemento de un patrón de entrada a cada uno de los perceptrones de entrada y otro patrón a cada uno de los perceptrones de la última capa de la red. Ésta, entonces, se entrena para que, dado un conjunto de patrones de entrada, se asocie un único patrón de salida; dado otro conjunto de patrones de entrada se asocie otro patrón único de salida, etc. La idea es que una RN así entrenada exhiba *generalidad*, es decir, que no solamente reconozca los patrones con los que fue entrenada, sino otros que exhiban características de similitud con el conjunto de entrenamiento. Esta generalidad implica la incorporación de la "esencia" de los patrones que se desea reconocer a la estructura misma de la red vía los pesos w_{ij} del arreglo y nuestro principal problema es cómo encontrar los pesos para lograr dicha plasticidad. El desarrollo del algoritmo llamado de retropropagación para determinar los pesos en dichos arreglos, ha hecho que estas redes sean las más usadas entre los investigadores y usuarios de redes neuronales.

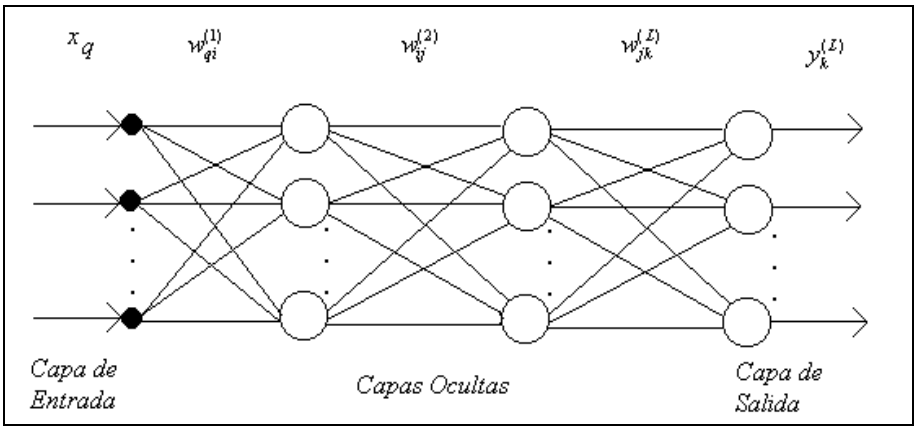


FIGURA F.E.1.B: *El perceptrón multicapa.*

ALGORITMO DE RETROPROPAGACIÓN

1. Inicialice los pesos (w_{ij}) con pequeños valores aleatorios.
2. Aleatoriamente seleccione un patrón de entrada $\boldsymbol{x}^{(u)}$.
3. Propague la señal hacia delante a través de la red.
4. Calcule δ_i^L en la capa de salida ($o_i = y_i^L$), en donde:

$$\delta_i^L = g'(h_i^L)[d_i^u - y_i^L]$$

   h_i^l representa la entrada neta a la *i*-ésima unidad de la *l*-ésima capa y *g'* es la derivada de la función de activación *g*; d_i^u representa la salida deseada.
5. Calcule las deltas para las capas precedentes por medio de la propagación de los errores hacia atrás;

$$\delta_i^l = g'(h_i^l)\sum_j w_{ij}^{l+1}$$

   para $l = (L-1), \ldots, 1$; *g'(h)* es la derivada de la función de activación.
6. Actualice los pesos usando

$$\Delta w_{ji}^l = \eta \delta_i^l y_j^{l-1};\ 0 < \eta \leq 1$$

7. Vaya al paso 2 y repita para el siguiente patrón hasta que el error en la capa de salida esté por debajo de un valor específico, o se alcance un número máximo de iteraciones.

En este algoritmo es importante tener una función de activación derivable. Por ello, la función escalón de la figura F.E.1.A suele reemplazarse por la función sigmoide $f(w) = (1+e^{-w})^{-1}$ cuya gráfica se muestra en la figura F.E.1.C.

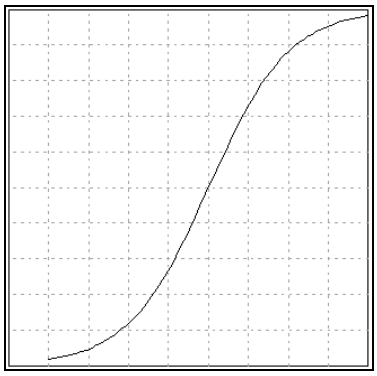


FIGURA F.E.1.C: *Función sigmoide.*

Considere el problema de encontrar el conjunto de valores que optimicen el comportamiento de la red neuronal esquematizada en la figura F.E.1.B. Suponemos que existe un conjunto de patrones de entrada y que, para cada uno de ellos, existe un patrón de salida "correcto". Por ejemplo, supongamos que un conjunto de patrones está constituido por las siguientes representaciones de las letras A, B, C, D:

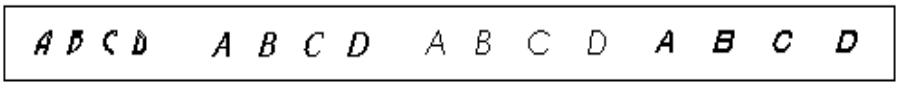


FIGURA F.E.1.D: *Distintos tipos de fuentes.*

Lo que deseamos es que para cualquiera de los diversos tipos (por ejemplo, las varias A's) se cumpla que la salida sea la letra "A". De igual manera, para las letras "B", "C", "D", etc., deberá cumplirse que, independientemente del tipo (ilustrado en la figura F.E.1.D) la salida corresponda con la letra deseada.

Para efectos de este ejercicio consideramos que existen otros dos conjuntos de neuronas similares al ilustrado, pero en planos ortogonales al papel, formando un "cubo" de i x j x k elementos (en la figura i = j = k, es decir, el cubo es de 3 x 3 x 3 elementos). Este arreglo se ilustra en la figura F.E.1.E, en la cual el patrón a reconocer se ejemplifica con los cuadrados del plano frontal, mientras que las neuronas corresponden a los cuadrados de los planos posteriores (en blanco). Un posible arreglo de 1 patrón de entrada, un patrón de salida y una red de tres capas se ilustra en la figura F.E.1.F.

A cada una de las neuronas (representada por un círculo vacío en F.E.1.B) le corresponde un conjunto de "pesos". Una neurona n_{ij} recibe *ijk* entradas en la capa *l*: una de cada uno de los *ijk* elementos que representan un patrón visual, como se ilustra en la figura F.E.1.F.

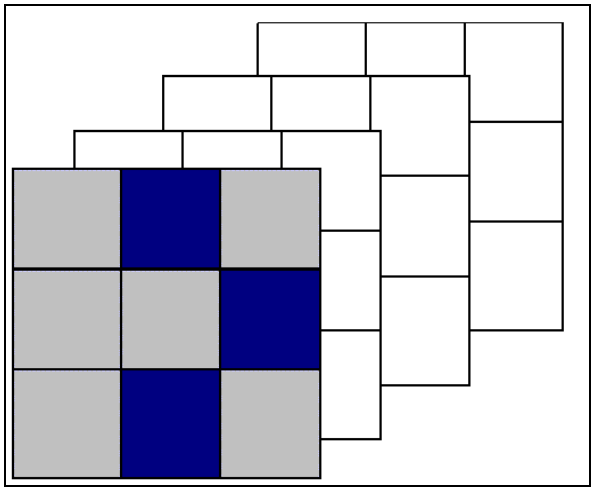


FIGURA F.E.1.E: *Arreglo de neuronas en 3D.*

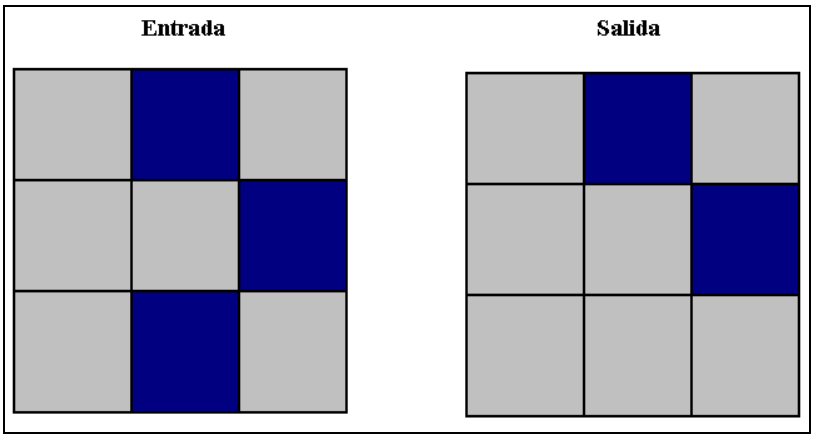


FIGURA F.E.1.F: *Patrones de entrada/salida.*

En el algoritmo de retropropagación está tácita la aceptación de una métrica de minimización de distancias. Además, al usar la primera derivada se logra una minimización de máxima pendiente. Asimismo, existe la necesidad de manejar una función de excitación que sea derivable.

Normalmente, ninguna de las dos premisas anteriores se cuestiona cuando se hace el análisis de RNs, como la descrita porque el algoritmo ya mencionado es aceptable y porque no hay buenas alternativas. Sin embargo, aquí proponemos una alternativa basada en el uso de AGs. Esta alternativa no se restringe a una métrica preseleccionada ni requiere una función de excitación derivable.

El entrenamiento de la red neuronal usando AGs es como sigue:

a. Especifique un número arbitrario de RNs (ρ) que considerará la élite del sistema.
b. Especifique un número arbitrario de RNs (μ) de dónde seleccionará los ρ miembros de la élite ($\mu > \rho$).
c. Especifique un número arbitrario de patrones de entrenamiento (π) que servirá para caracterizar a cada RN en la élite.
d. Haga $i \leftarrow 1$.
e. Haga $j \leftarrow 1$.
f. Elija un conjunto arbitrario de patrones de entrada C_j.
g. Elija un conjunto arbitrario de pesos W_j $(-1 \leq W_j \leq +1)$.
h. Evalúe el número de elementos de la capa de salida que coinciden con los de cada uno de los patrones de entrenamiento.
i. Obtenga el promedio de aciertos por patrón. A ese promedio llámele D_j.
j. Haga $j \leftarrow j+1$.
k. Si $j < \pi$ continúe con el paso (f); de lo contrario continúe con el paso (l).

l. Haga $i \leftarrow i+1$.
m. Si $i < \mu$ continúe con el paso (e); de lo contrario continúe con el paso (n).
n. Seleccione las mejores ρ redes neuronales.
o. $i \leftarrow 1$.
p. Elija un conjunto arbitrario de patrones de entrada.
q. Elija un conjunto arbitrario de pesos.
r. Aplique un AG que optimice el comportamiento de la red.
s. Elija un conjunto arbitrario de patrones.
t. Aplique a la élite y a la red neuronal.
u. Califique a la red en términos comparativos.
v. Si la red neuronal está dentro de las mejores ρ RNs, inclúyala en el grupo élite eliminando la peor RN.
w. $i \leftarrow i+1$.
x. Si $i < \rho$ ir al paso (p); de lo contrario continúe con (y).
y. Fin del algoritmo.

PROBLEMA

1. Discuta los beneficios y/o desventajas del esquema propuesto vs. el esquema tradicional. ¿Cómo se compara la estrategia de retropropagación vs. la estrategia genética? ¿Qué otras alternativas que usen AGs existen? (proponga al menos otra) ¿Por qué es interesante usar un AG coevolutivo?

AUTÓMATAS GENÉTICOS

2. ¿De qué manera cambiaría un autómata genético si en vez de usar máquinas de Turing (MT) se usaran autómatas de estados finitos?

3. ¿Cómo se lograría extender el método de predicción para lograr que el algoritmo ajuste varias secuencias de entrada con varias (correspondientes) secuencias de salida simultáneamente?

SISTEMAS CLASIFICADORES EVOLUTIVOS

4. Suponga que la apuesta efectiva de un clasificador es constante. ¿Cómo afectaría esta consideración el desempeño del clasificador?

5. Suponga que va a resolver el problema de manejar la apertura/cerradura de una válvula como la que se plantea en la figura F.V.4.C. Contraste los enfoques de Pitt y Michigan. ¿Cómo manejaría la posibilidad de que el conjunto de reglas (en el enfoque Pitt) sea de longitud variable?

6. Suponga que desea expresar el comportamiento en el tiempo de un sistema de reservación de boletos. Suponga que posee los datos históricos de los últimos 48 meses. ¿Cómo habría que ajustar los parámetros para que la demanda se exprese como una función de los últimos seis meses inmediatos anteriores?

7. En general, cuando los datos de tablas como la tabla T.V.5.A son de tamaños relativos muy distintos, es necesario normalizarlos. Es decir, es necesario hacer que todos ellos estén en un rango similar. Plantee una fórmula de uso general que le permita que todos los valores de la tabla estén en el intervalo [a,b] usando una transformación lineal.

V.7 PROGRAMACIÓN

1. Escriba un programa que entrene a una red neuronal sin apelar al algoritmo de retropropagación, usando un AG.

2. Escriba un programa que permita encontrar la mejor MT usando como entrada una cadena alfabética de longitud arbitraria y como salida otra cadena, también arbitraria. Utilice:

   a) Un algoritmo genético elitista.
   b) Un algoritmo genético ecléctico.

3. Comente acerca de las diferencias observadas.

4. Escriba un programa que implemente un SCE para resolver el problema número 5.

5. Escriba un programa que implemente un MABE para el problema anterior usando el enfoque de Pitt.

6. Resuelva el problema 6 (definido arriba) usando:

   a) Una red neuronal como la del problema de programación 1.
   b) Un algoritmo genético de orden libre.

7. Contraste los resultados de aplicar cada uno de los métodos.

V.8 REFERENCIAS

[1] Hofstadter, D., "Dilemmas for Superrational Thinkers, Leading Up to a Luring Lottery", *Methamagical Themas*, Bantam Books, 1985.

[2] Mitchell, M., "Using GAs to Evolve Strategies for the Prisoner's Dilemma", *Introduction to Genetic Algorithms*, MIT Press, 1996.

[3] Holland, J., *Hidden Order*, Addison-Wesley, 1995.

[4] Axelrod, R., *The Evolution of Cooperation*, Basic Books, 1984.

[5] Fogel, L. J., "Autonomous automata", *Industrial Research*, Vol. 4, 1962, pp. 14-19.

[6] Turing, A., "On Computable Numbers with Application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society*, Vol. 42, 1937, p. 230.

[7] Beckman, F., *Mathematical Foundations of Programming*, Addison-Wesley, 1981.

[8] De Jong, K., y Spears, W., *An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms*, kdejong@aic.gmu.edu, 1995.

[9] Goldberg, D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989, pp. 40-41.

[10] Wilcox, J., *Organizational Learning Within a Learning Classifier System*, University of Illinois at Urbana-Champaign, IlliGAL Report No. 95003, May, 1995.

[11] Kuri, A., "An Alternative Model of Genetic Algorithms as Learning Machines", *Fourth World Congress on Expert Systems*, March, 1998.

[12] De Jong, K. A., "Learning with Genetic Algorithms: An Overview", *Machine Learning*, Vol. 3, 1988, pp. 121-138.

[13] Hamming, R., *Introduction to Applied Numerical Analysis*, McGraw-Hill, 1971.

[14] Cheney, E. W., *Introduction to Approximation Theory*, McGraw-Hill Book Company, 1966, p. 39.

[15] Kuri, A., "Heuristic Techniques for the Generalization and Acceleration of an Algorithm for Multivariate Minimax Approximation", *AGROCIENCIA*, 1993, pp. 94-95.

[16] Jain, A. K., y M. Jianchang, "Artificial Neural Networks: A Tutorial", *Computer*, Vol. 29, No. 3, marzo, 1996, pp. 31-44.

ÍNDICE GENERAL

I. Introducción 19

I.1 La naturaleza como optimizadora 19
I.2 Un poco de Biología 21
I.3 Algoritmos genéticos 24
I.3.1 Codificación del dominio 24
I.3.2 Evaluación de la población 25
I.3.3 Selección 26
I.3.4 Cruzamiento 27
I.3.5 Mutación 28
I.3.6 El algoritmo genético simple (AGS) 28
I.3.7 Un ejemplo 31
I.4 Panorama de otros métodos de optimización 37
I.4.1 Métodos tradicionales 37
I.4.1.1 Método de Newton 37
I.4.1.2 Búsqueda de Fibonacci 39
I.4.1.3 Ascenso de máxima pendiente (steepest ascent) 41
I.4.1.4 Simplex 43
I.4.2 Métodos heurísticos 47
I.4.2.1 Búsqueda tabú 47
I.4.2.2 Recocido simulado 49
I.4.3 Comparaciones 51
I.5 Ejercicios 53
I.6 Programación 55
I.7 Referencias 56

II. Fundamentos matemáticos 53

II.1 Terminología 53
II.1.1 Función de adaptación 53
II.1.2 Esquemas 53
II.2 El teorema del esquema 54
II.2.1 Selección 54
II.2.2 Cruzamiento 56
II.2.3 Mutación 57
II.2.4 Tipos de cruzamiento 58
II.2.4.1 Cruzamiento de un punto 59
II.2.4.2 Cruzamiento de dos puntos 60
II.2.4.3 Cruzamiento uniforme 60
II.3 Críticas 61
II.4 Paralelismo implícito y la HBC 62
II.5 Exploración y explotación 63
II.6 Engaños y resultados inesperados 67
II.6.1 Problemas engañosos mínimos 67

II.6.1.1 Los tipos de problemas engañosos mínimos 67
II.6.1.2 Análisis de los problemas engañosos 71
II.6.2 Caminos regios 76
II.6.2.1 Definición del problema 76
II.6.2.1 Escaladores 80
Resultados 82
II.7 Ejercicios 83
II.8 Programación 85
II.9 Referencias 86

III. Variaciones sobre un tema de Holland 87

III.1 Caso de Estudio: El Problema del Agente Viajero 87
III.1.1 Codificación del dominio 87
III.1.1.1 Codificación binaria 87
III.1.1.2 Codificación no binaria 89
III.1.2 Operadores 90
III.1.2.1 Operador de cruza 91
III.1.2.2 Cruzamiento uniforme ordenado 91
III.1.2.3 Mutación 92
III.1.3 La función de adaptación 93
III.1.4 Síntesis 94
III.2 Caso de estudio: optimización de una función 95
III.2.1 Codificación del dominio 95
III.2.1.1 Codificación en punto fijo 95
III.2.1.2 Codificación binaria pesada 96
III.2.1.3 Codificación binaria de Gray 97
III.2.1.4 Codificación en punto flotante 97
III.2.1.5 ¿Cuál es la mejor opción? 99
III.2.2 Tres algoritmos 101
III.2.3 Experimentos 103
Resultados 105
III.3 Ejercicios 111
III.4 Programación 112
III.5 Referencias 112

IV. Algoritmos genéticos no convencionales 113

IV.1 Introducción 113
IV.2 Un algoritmo genético idealizado 113
IV.2.1 Conclusiones 115
IV.3 Modelos de cadenas de Markov 115
IV.3.1 Conclusiones 117
IV.4 Modelos no convencionales 118
IV.4.1 Elitismo 118
IV.4.2 Selección determinística 119
IV.4.3 Modelo de Nietzsche 120
IV.4.4 Modelo de Vasconcelos 120
IV.4.5 Autoadaptación 121
IV.4.5.1 Algoritmo genético autoadaptable individual 122

IV.4.5.2 Algoritmo genético autoadaptable poblacional 123
IV.4.6 Un algoritmo genético ecléctico 125
IV.4.6.1 Selección, elitismo y autoadaptación....126
IV.4.6.2 Escalador adaptivo....128
IV.5 Ejercicios 131
IV.6 Programación....132
Representación de números en un genoma....133
IV.7 Referencias 134

V. Algoritmos genéticos aplicados al aprendizaje....137

V.1 Inteligencia y aprendizaje 137
V.2 Algoritmos genéticos coevolutivos....137
V.2.1 Algoritmo de coevolución....138
V.2.2 El dilema del prisionero....139
V.2.3 Algoritmo DPI....141
V.3 Autómatas genéticos....142
V.3.1 Máquinas de Turing 143
V.3.2 Viabilidad de un autómata genético....145
V.3.3 Algunas consideraciones en referencia al modelo planteado 149
V.3.3.1 Determinación de la validez del modelo....149
V.3.3.2 Generalidad del modelo 151
V.3.3.3 Probabilidades de cruzamiento 152
V.3.3.4 Coeficiente de correlación predictiva 152
V.4 Sistemas clasificadores evolutivos....154
V.4.1 Sistema de reglas de producción....155
V.4.2 Asignación de crédito....159
V.4.3 Algoritmo de descubrimiento de clasificadores 162
V.4.4 Jerarquías por omisión 163
V.4.4.1 Ejemplo de jerarquías por omisión 164
V.5 Ajuste paramétrico de orden libre....167
V.5.1 Fundamentos matemáticos....168
V.5.1.1 Solución al conjunto M....168
V.5.1.2 Un ejemplo numérico....172
V.5.1.3 Solución al conjunto N....174
V.5.1.4 Algoritmo de ascenso....174
V.5.2 Solución de sistemas singulares....176
V.5.3 Solución de sistemas completos 182
V.5.4 Algoritmo genético de orden....183
V.6 Ejercicios....187
V.7 Programación....194
V.8 Referencias 195