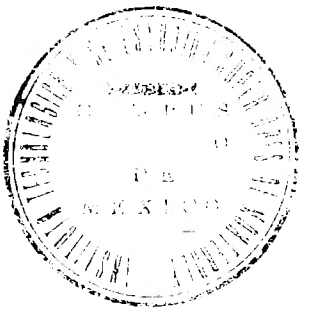
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES
DE MONTERREY

ALGORITMOS GENÉTICOS PARA LA GENERACIÓN AUTOMÁTICA DE MÁQUINAS DE TURING



Trabajo de investigación que, para obtener el grado de

MAESTRO EN CIENCIAS COMPUTACIONALES

Presentó Lic. Edgar Emmanuel Vallejo Clemente

Siendo integrado el jurado por

Dr. Oscar Chavoya Aceves

Dr. Martín López Morales

Dr. Jesús Sánchez Velázquez

Mayo 1994

Contenido

Capítulo 1

Introducción

1.1 Resumen.

La máquina de Turing es un modelo de la computación capaz de representar cualquier solución algorítmica a un problema. El presente trabajo de investigación propone un modelo de entrenamiento de máquinas de Turing mediante el uso de algoritmos genéticos. Dichos algoritmos llevan a cabo la búsqueda de un elemento particular, a través de la aplicación de mecanismos de selección natural y de la genética a una población de máquinas de Turing aleatoriamente generada, cuyos elementos se representan mediante la codificación de la forma tabular correspondiente a la función de transición δ.

El sistema de generación automática de máquinas de Turing puede ser considerado como un sistema de programación automática, ya que permite la generación de un procedimiento algorítmico que produce una respuesta deseada para entradas particulares; y que puede ser empleado en la solución de una gran variedad de problemas que se plantean en diversas áreas de las ciencias computacionales.

Los resultados experimentales demuestran que mediante la utilización de este modelo, es posible encontrar procedimientos automáticos para la determinación de membresía a lenguajes formales que definen conjuntos de secuencias (de DNA) del virus de HIV, así como llevar a cabo la alineación (localización de segmentos comunes) de las secuencias.

1.2 Justificación.

Existe una variedad de problemas en las áreas de Inteligencia Artificial, Aprendizaje de Máquina (Machine Learning), y Procesamiento Simbólico que se plantean como la búsqueda de un algoritmo que produce una salida deseada para entradas particulares. El proceso de resolver estos problemas se reduce a encontrar en el espacio de búsqueda de procedimientos algorítmicos, un elemento particularmente adecuado para la solución del problema.

Una pregunta central en las ciencias computacionales es "¿ Cómo puede una computadora aprender a programarse a sí misma para resolver problemas ?" La necesidad de automatizar el desarrollo de procedimientos algorítmicos, concretamente, el desarrollo de sistemas computacionales, ha sido reconocida desde el advenimiento de la computadora digital. El campo de programación automática estudia programas de computadora que sintetizan nuevos y diferentes programas, o que se modifican y perfeccionan a sí mismos.

Los investigadores responsables de llevar a cabo los primeros trabajos en programación automática (1958-1970), estimaron tener éxito en la creación de programas deseados a partir de la ejecución de programas que se modifican a sí mismos utilizando un procedimiento aleatorio de generación y evaluación. Esta hipótesis fracasó debido a que la necesidad de recursos de cómputo para sintetizar nuevos programas parecía incrementarse exponencialmente en relación a su tamaño. Así que, todos los intentos de hacer evolucionar programas fallaron a causa de la explosión combinatoria.

Tal es el caso del trabajo realizado por R.M. Friedberg [Frie59], investigador de IBM. Su sistema llevaba a cabo una búsqueda a través del espacio de todos los programas codificados en un lenguaje de máquina de 64 instrucciones. Este programa reemplazaba todas las instrucciones (una a una) intentando localizar un máximo en el desempeño, repitiendo este procedimiento una y otra vez, cientos de veces por segundo en una computadora IBM 704. Cuando el programa deseado contenía un par de instrucciones (por ejemplo sumar dos números de un bit), fueron necesarias cientos de miles de generaciones para hacer evolucionar dicho programa. Cuando el programa deseado era mayor, digamos cinco o seis líneas de código, raramente apareció después de millones de generaciones.

Además, el procedimiento de búsqueda de la "escalada de la colina" no exhibió mejor desempeño que la simple caminata al azar. Friedberg construyó un sistema que generaba nuevos programas desde "scratch" en cada generación, ignorando por completo el diseño de sus "antecesores", no importando que tan cercano fuera su comportamiento al del programa deseado. Este programa superó en desempeño al programa que empleaba el procedimiento de escalada de la colina gradual.

La inestabilidad de los programas codificados en lenguaje de máquina fue la principal desventaja en relación con el desarrollo de los sistemas de programación automática; esto es, una pequeña modificación a la estructura lógica del programa y/o al contenido de ciertas localidades de memoria alteraba dramáticamente la función computada por el programa.

Con la finalidad de superar el problema anterior, Fogel, Owens, y Walsh [Foge66] propusieron técnicas de evolución de programas en otro nivel. El sistema desarrollado por ellos llevaba a cabo tareas de predicción de secuencias de símbolos mediante una búsqueda en el espacio de máquinas de estados finitos (máquinas de Mealy). Las técnicas de evolución de programas aplicaban operadores de mutación a la gráfica de transiciones de la máquina de estados finitos, esto es, re-direccionaba arcos, añadía nodos, re-etiquetaba arcos, etc. En cada generación el programa seleccionaba un mecanismo de mutación para alterar a la máquina que mostraba el mejor desempeño hasta el momento. De la misma manera, el procedimiento de la "escalada de la colina" basado en la mutación aleatoria parecía desarrollarse muy lentamente, quedando atrapado en máximos locales.

Durante las décadas de 1970's y 1980's se alcanzaron progresos significativos en el campo de programación automática en base a la creación de sistemas provistos de una gran cantidad de conocimiento acerca de la programación en general y del campo específico en el que operarían los programas sintetizados. Al utilizar dicho conocimiento para restringir y guiar la búsqueda, los programas finalmente comenzaron a generar nuevos programas y modificarse a sí mismos exitosamente. Sin embargo, en la mayoría de los sistemas de programación automática, el conocimiento fue provisto por expertos humanos, sólo en el caso de algunos sistemas como AM [Lena76] y EURISKO [Lena83] éste fue descubierto automáticamente.

La iniciativa de utilizar sistemas basados en conocimiento (sistemas expertos) como mecanismo fundamental en los sistemas de programación automática, se debe principalmente a que la maquinaria necesaria para la adquisición y explotación de conocimiento es relativamente sencilla: acumular una base de información empírica y realizar generalizaciones inductivas simples a partir de ésta. La primera actividad requiere de algún tipo de memoria, la segunda de habilidad en el reconocimiento de patrones. Los sistemas de programación lógica inductiva [Ramí94] proveen un mecanismo para llevar a cabo el descubrimiento automático de conocimiento.

Se conoce que procesos similares de almacenamiento y reconocimiento ocurren en los seres humanos a nivel molecular (información confiable almacenada en los ácidos nucleicos, un sistema de reconocimiento de patrones confiable del tRNA al mRNA en los ribosomas) y de igual manera en niveles más altos (memorización en el cerebro, reconocimiento de patrones por el sistema inmune). Esto nos lleva a hipotetizar que el proceso de generación de mutaciones en los seres vivos puede ser considerablemente no aleatorio. Así que, en lugar de un proceso aleatorio de generación y evaluación, el mecanismo dominante de la evolución en organismos avanzados debe ser un proceso "sofisticado" de generación y evaluación.

En 1975 John Holland, en su libro *Adaptation in Natural and Artificial Systems* [Holl75] describe cómo el proceso de evolución en la naturaleza puede ser aplicado a sistemas artificiales. En particular, Los algoritmos genéticos de Holland son procedimientos matemáticos altamente paralelos que transforman a una población de objetos matemáticos individuales (generalmente cadenas de caracteres de longitud fija) en una nueva población, mediante el uso de operadores que imitan los procesos de reproducción (principio de Darwin de supervivencia del mejor), recombinación sexual (cruzamiento), y mutación aleatoria (en pequeñas cantidades). Los algoritmos genéticos fueron empleados por Holland [Holl86], Wilson [Wils87] y Forrest [Forr91] en el desarrollo de sistemas de programación automática.

A principios de la presente década, J.R. Koza [Koza92] presenta el paradigma de programación genética, metodología que provee una manera para llevar a cabo la búsqueda en el espacio de programas de computadora. En el paradigma de programación genética, poblaciones de programas de computadora son genéticamente reproducidos utilizando el principio de Darwin de supervivencia del mejor individuo y un operador genético de cruzamiento (recombinación

sexual). Koza propone una estructura jerárquica (árboles de expresión que representan instrucciones de lenguaje LISP) como esquema de representación, ya que permite cierta flexibilidad en la forma, tamaño y complejidad estructural de la solución.

Koza argumenta que la representación de la solución del problema por medio de cadenas de caracteres de longitud fija (como la que se emplea en algoritmos genéticos) puede significar una importante limitación en la solución del problema: "*La representación es un aspecto clave en el funcionamiento del algoritmo genético ya que los algoritmos genéticos manipulan directamente la representación codificada del problema y debido a que el esquema de representación puede limitar severamente la ventana a través de la cual el sistema observa su propio mundo.*"

"*Los esquemas de representación basados en cadenas de caracteres de longitud fija no proveen la estructura jerárquica central para la organización de programas de computadora (en programas y subrutinas), o de la incorporación de iteración o recursión cuando éstas sean inherentemente necesarias para la solución del problema.*"

Sin embargo, el paradigma de programación genética impone restricciones a su propio esquema de representación, mediante el establecimiento de un límite de profundidad en la estructura jerárquica empleada para representar la solución del problema (durante el proceso de recombinación sexual). De esta manera, limita el número de estados internos que la solución puede asumir.

Por otro lado, debido a las restricciones impuestas por la definición del lenguaje LISP y por la naturaleza del problema, en la mayoría de los casos prácticos será necesario efectuar el análisis semántico de cada una las instrucciones que se generan durante el proceso de evolución de programas, lo que representa un costo importante en recursos de cómputo.

Es evidente que las limitaciones de los sistemas de programación automática desarrollados anteriormente, se deben principalmente a las restricciones que impone el modelo de computación y no el esquema de representación. Por lo que es necesario elegir un modelo de cómputo lo más general posible, cuyo poder exceda a los modelos anteriores, lo que nos lleva a la consideración de la máquina de Turing.

La máquina de Turing es un modelo abstracto de la computación, introducido por Allan Turing en 1936 [Turi36] cuyo poder de cómputo representa el límite fundamental de la computadora digital. De hecho, cualquier solución algorítmica a un problema puede ser representada mediante el programa de instrucciones de una máquina de Turing incluyendo diseños "modulares" o "top-down", es decir, la máquina de Turing puede simular cualquier tipo de subrutina que pueda encontrarse en los lenguajes de programación, incluyendo procedimientos recursivos o cualquier mecanismo conocido de paso de parámetros.

La existencia de la máquina de Turing universal demuestra que existe un cierto nivel mínimo de complejidad de los sistemas computacionales que es suficiente para llevar a cabo cualquier computación algorítmica. Minsky [Mins67] ha demostrado cómo los requerimientos esenciales para cualquier procedimiento efectivo son alcanzados por medio de sistemas formales muy simples, incluyendo una máquina de Turing en la que la cardinalidad del conjunto de estados es siete, con un alfabeto de cinta que consta de cuatro símbolos.

En base a lo anterior se propone un modelo de entrenamiento de máquinas de Turing mediante algoritmos genéticos, considerando que el esquema de representación tenga la habilidad de caracterizar (como mínimo) máquinas de Turing de siete estados con alfabeto de cinta de cuatro símbolos.

Cabe mencionar, que es posible transportar la simulación del sistema de generación automática de máquinas de Turing más allá de los límites de la computadora digital, ya que se conoce que es posible simular todas las máquinas de Turing por medio de redes neuronales completamente interconectadas (redes de Hopfield), en tiempo lineal. En particular, es posible construir una red con aproximadamente 1,000 procesadores capaz de computar una función universal parcialmente-recursiva [Sieg91].

El modelo de generación automática de máquinas de Turing puede ser empleado en la solución de una gran variedad de problemas (como por ejemplo calcular cualquier función recursiva), sin embargo, se ha decidido aplicar el modelo al análisis de las secuencias de DNA (que se generan a partir de un alfabeto de cuatro símbolos) correspondientes a las variedades africana y americana del virus de HIV, lo que permite encontrar lenguajes formales que definen a las secuencias anteriores, así como localizar regularidades en las

mismas. Lo anterior puede representar una herramienta importante de análisis para el campo de la biología molecular.

El desarrollo del modelo se apoya en la teoría de lenguajes formales y en los sistemas evolutivos, por lo que serán expuestos detalladamente en los capítulos 2 y 3 respectivamente.

Capítulo 2

Teoría de Automátas y Lenguajes Formales

2.1 Introducción.

El aspecto teórico de las ciencias computacionales ha evolucionado a partir de tres disciplinas: las matemáticas, la ingeniería, y la lingüística. Las raíces matemáticas de las ciencias computacionales datan de la década de los treintas cuando el trabajo de Turing expone los límites fundamentales de la computación mecánica. Los orígenes en ingeniería comenzaron con la observación de Shannon, de que las funciones de una red de switcheo podían ser representadas por medio de la notación simbólica del álgebra booleana. La contribución de la lingüística a las ciencias computacionales se presenta mediante la caracterización de Chomsky en relación a las gramáticas y lenguajes formales. El interés en la especificación formal de lenguajes de programación motivó a científicos de computación a estudiar el trabajo de Chomsky y construir una importante teoría de lenguajes generados artificialmente.

Con el advenimiento de la computadora digital, la curiosidad intelectual del hombre se ha enfocado a la utilización de la información. El entendimiento de las máquinas que procesan información y del cómputo que pueden efectuar, ha motivado a personas de diversos campos. El campo de las ciencias computacionales ha logrado avances significativos en sus estudios.

El estrepitoso incremento en la velocidad de procesamiento y en la capacidad de almacenamiento de la computadora digital, ha dado origen a preguntas relacionadas con los límites de su poder: ¿ Cuáles son, si existen, las fronteras de las tareas que puede realizar ? Con el fin de entender sus limitaciones se crean dos abstracciones de estos dispositivos, las máquinas abstractas, o *autómatas*, y los lenguajes abstractos. Los autómatas han sido utilizados para modelar una variedad de sistemas, desde circuitos digitales hasta el sistema nervioso humano. Los lenguajes abstractos han contribuído en la modelación de lenguajes de programación y subconjuntos simples del lenguaje natural.

La noción de una máquina abstracta, o autómata, fue concebida por Alan M. Turing con el propósito de explotar los límites de la habilidad del ser humano en formalizar métodos para la solución de problemas. Propuso que el término *algoritmo*, debía considerarse como un procedimiento que puede ser llevado a cabo por un tipo específico de autómata llamado *máquina de Turing*. El estudio de este dispositivo dio como resultado un persuasivo argumento llamado *tesis de Church: El poder de cómputo de una máquina de Turing representa un límite fundamental en la capacidad de los dispositivos de cómputo*.

Los lingüistas han buscado por mucho tiempo las gramáticas formales adecuadas de los lenguajes naturales con el fin de explicar la remarcable habilidad humana en interpretar enunciados que no han escuchado antes. Mientras que sus estudios no han tenido mucho éxito, sí han provisto de un conocimiento considerable acerca de diversas clases de lenguajes abstractos. Tomando en cuenta la relación entre los lenguajes y los autómatas, se ha encontrado que es útil considerar a un lenguaje como un conjunto de *enunciados* que satisfacen ciertas propiedades o reglas de construcción.

La relación entre los lenguajes abstractos y los autómatas surge con los escritos de Kleene, en los que caracteriza a los lenguajes en los cuales la membresía de un enunciado puede ser decidida por una máquina de estados finitos. Desde entonces, mucho se ha aprendido en relación a la habilidad de los autómatas para reconocer y generar lenguajes abstractos. El trabajo de Noam Chomsky establece la relación entre las clases de lenguajes abstractos y una jerarquía de tipos de máquinas de estados finitos que se muestra en la figura 2.1.

En este apareamiento de máquinas abstractas y clases de lenguajes, se puede esperar que un modelo de máquina más poderoso se encuentre asociado con una clase menos restringida de lenguaje, lo cual sucede en la práctica.

La jerarquía comienza con las máquinas de estados finitos y los lenguajes regulares, progresando hacia máquinas más poderosas y lenguajes menos restringidos. En cada nivel, existen problemas o lenguajes que exceden el poder de la máquina de estados finitos correspondiente. Excepto para el nivel de la máquina de Turing, existe una generalización natural en cada caso que nos lleva al siguiente nivel más alto de máquinas y lenguajes. Para las máquina de Turing, no se ha establecido generalización alguna que sobrepase su capacidad. Aun así, existe una conocida clase de problemas cuya naturaleza rebasa el poder de las máquinas de Turing (funciones y lenguajes no computables).

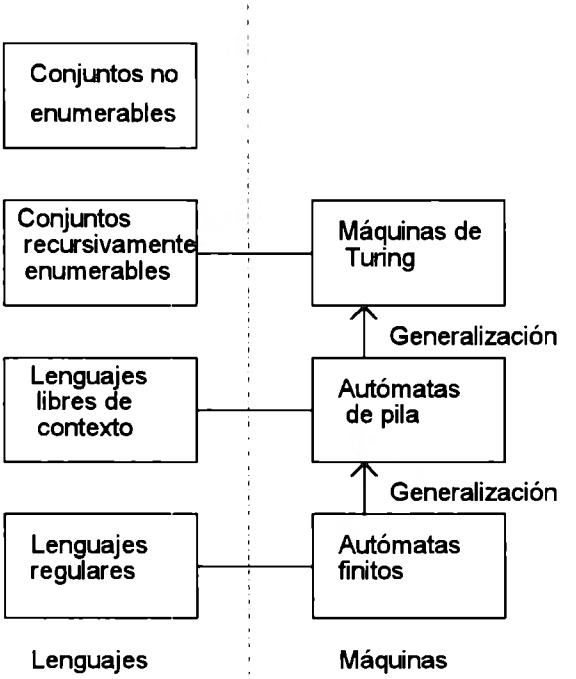


Figura 2.1. Jerarquía de máquinas y lenguajes

2.1 Definiciones básicas.

Un *alfabeto* consiste de un conjunto finito no vacío de elementos atómicos llamados símbolos y es denotado por Σ. Las letras minúsculas *a*, *b*, *c*, *d*,..., son utilizadas para representar a los elementos de dicho conjunto. Un *string* es una tupla de *n* elementos de Σ. Los strings de un alfabeto son representados por las últimas letras del alfabeto latino. En particular *u*, *v*, *w*, *x*, *y*, *z*, denotan strings. La longitud de un string *w*, denotada por $|w|$ es el número de símbolos que

componen el string. El *string nulo*, denotado por λ, es el string cuya longitud es cero. Esto es, $|\lambda| = 0$.

Un *prefijo* de un string w, es cualquier número de símbolos menor o igual a $|w|$, que se encuentran al principio del string y un sufijo de w es cualquier número de símbolos menor o igual a $|w|$ que se encuentran al final. Cualquier prefijo o sufijo con longitud menor a $|w|$ es llamado prefijo o sufijo propio de w.

El conjunto que contiene a todos los strings que se pueden formar con el alfabeto se denota por Σ^*. La operación fundamental de los strings es la *concatenación*, una operación binaria en el conjunto Σ^*:

$$m: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$$

La concatenación de los strings w y x es la secuencia de símbolos formada al yuxtaponer el string w seguido del string x:

$$m:(w,x) = a_1 a_2 \ldots a_m b_1 b_2 \ldots b_n$$

donde:

$$w = a_1 a_2 \ldots a_m, \text{ y } x = b_1 b_2 \ldots b_3$$

Para representar la concatenación de strings se emplea la siguiente notación: $w \cdot x$, o simplemente wx. La operación de concatenación induce las siguientes propiedades:

1. $\forall\, w,x,y \in \Sigma^*, w \cdot (x \cdot y) = (w \cdot x) \cdot y$. Esto es, la concatenación es asociativa.

2. $\forall\, w \in \Sigma^*, w \cdot \lambda = \lambda \cdot w = w$. Esto es, existe un elemento identidad para la concatenación.

3. No se cumple necesariamente que $\forall w,x \in \Sigma^*, w \cdot x = x \cdot w$. Así que la concatenación no es conmutativa.

4. $\forall\, w,x \in \Sigma^*, |w \cdot x| = |w| + |x|$. Esto es, la longitud de la concatenación está dada por la suma de las longitudes de los strings que la componen.

5. $\forall w, x \in \Sigma^*, w \cdot x \in \Sigma^*$. Así que el conjunto Σ^* se encuentra cerrado bajo la concatenación.

Un *lenguaje formal* es cualquier subconjunto de Σ^*. También son lenguajes, el conjunto vacío, $\varnothing$, y el conjunto que contiene al string nulo $\{\lambda\}$. Las operaciones básicas sobre los lenguajes incluyen las operaciones de la teoría de conjuntos: unión, intersección, complemento. Se define la concatenación de dos lenguajes L y M como el conjunto de todos los strings que resultan al concatenar cada uno de los strings de M a cada uno de los strings de L. Esto es:

$$L \cdot M = \{w \cdot x | \, w \in L, x \in M\}$$

De la misma manera que la concatenación de strings, la concatenación de lenguajes constituye una estructura de semigrupo.

Sea Σ un alfabeto, sea Σ^* es universo de los strings generados a partir de Σ. Para el lenguaje $L \in \Sigma^*$, se definen inductivamente el conjunto $L^k, k \geq 0$, como:

$$L^0 = \lambda, \text{ y } L^{k+1} = L^k \cdot L$$

Cada uno de los conjuntos L^k es un subconjunto de Σ^* debido a que Σ^* se encuentra cerrado por la operación de concatenación. La unión de estos conjuntos:

$$L^* \equiv \bigcup_{i=0}^{\infty} L^i$$

Es también un subconjunto de Σ^* llamado la cerradura de L. El operador unario * es conocido como el operador de Kleene. La cerradura de un lenguaje contiene a cada string que puede formarse por medio de la concatenación de un número arbitrario de strings tomados del lenguaje.

2. Máquinas de Estados Finitos.

Las computadoras, circuitos de conmutación, sistemas de comunicación o cualquier máquina de información en general, se construyen mediante la interconexión de una gran cantidad de componentes simples. Sólo un número finito de componentes físicos pueden estar contenidos en un volumen específico. Los *símbolos* que se utilizan para señalar eventos entre componentes son comúnmente representados por valores de cantidades físicas como posiciones mecánicas, voltajes, temperaturas, presiones. Debido a que ninguna señal puede ser medida con gran exactitud dentro de pequeñas tolerancias, los diseñadores de dispositivos físicos están limitados a un pequeño número de valores fácilmente distinguibles, tanto para las señales, como para las condiciones internas (estados) de los dispositivos. De esta manera, la realidad física impone una concepción finita a los sistemas físicos.

Una señal o estado requieren, por lo general, de un pequeño intervalo de medición. Debido a lo anterior, se sigue que únicamente un número finito de operaciones puede ser desempeñadas confiablemente en una cantidad finita de tiempo. La realidad física impone una naturaleza discreta a los sistemas físicos, no sólo en los símbolos que pueden procesar o los estados que pueden asumir, sino que también en las ocasiones en las cuales puede existir un cambio de estado.

La tendencia natural a descomponer las soluciones de un problema en secuencias de pasos se encuentra manifestado en el modelo de máquina de estados finitos mediante la acción secuencial. Esta claro que diseñar una máquina para operar un paso a la vez es la manera más simple conocida de hacer su comportamiento determinista.

Las propiedades de un sistema finito, discreto, determinista y la acción secuencial personifican al modelo de máquina de estados finitos. No obstante sus fuertes raíces en los fenómenos físicos, las máquinas de estados finitos no se encuentran limitadas a modelar procesos físicos. Son frecuentemente aplicadas en procedimientos de procesamiento de información, tales como la codificación-decodificación de mensajes o en el análisis de léxico y sintáctico de programas de computación. Además juegan un papel central en modelos más poderosos de computación, donde aparecen como la unidad de control de una máquina abstracta que tiene acceso a un medio de almacenamiento con capacidad ilimitada (máquina de Turing).

Existe una multitud de procesos de información en los cuales no corresponde un modelo de estados finitos. A pesar de sus limitaciones, este modelo juega un papel importante en la teoría de la computación como una medida absoluta de la complejidad de procesos computacionales.

La relación entre los lenguajes formales y las máquinas de estados finitos se establece mediante el estudio de dos tipos de autómatas: aceptadores y traductores de lenguajes. La figura 2.2 muestra un aceptador de lenguaje. Los símbolos del string $s(1)s(2)....s(t)$ se presentan secuencialmente a la máquina **M**, la cual responde a cada símbolo con un señal binaria. Los símbolos son tomados del alfabeto de entrada de **M**, y se asume que **M** se encuentra en un estado predeterminado de existencia (estado inicial) antes que el string de entrada le sea presentado. Adicionalmente, se supone que el comportamiento de la máquina es determinista, es decir, su respuesta depende únicamente de su estado inicial y de la secuencia de entrada presentada.

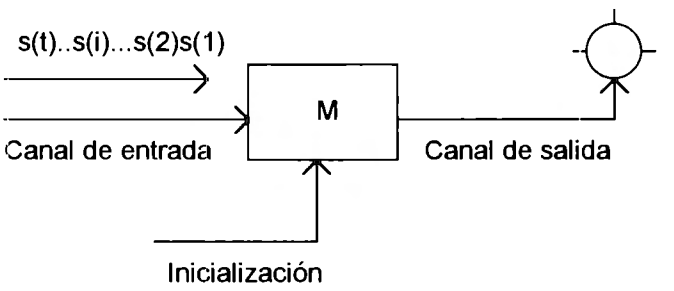


Figura 2.2. Aceptador de lenguaje

La máquina **M**, divide a todos los posibles strings de entrada en dos clases, de acuerdo a la señal que presenta el canal de salida inmediatamente después que se ha presentado el último símbolo de la secuencia: Los strings a los cuales **M** responde con una señal de encendido y los strings a los que **M** responde con una señal de apagado. Se dice que **M** acepta a cada secuencia perteneciente a la primera clase, y rechaza a cada secuencia de la segunda. Si **M** acepta a los strings de un lenguaje **L**, se dice que **M** reconoce a **L**.

La figura 2.3 muestra un traductor de lenguaje. El traductor **M** genera la secuencia $r(1)r(2)...r(n)$ en respuesta a la secuencia $s(1)s(2)....s(m)$ que se le presenta a la entrada. Dicha máquina, que exhibe un comportamiento determinista, traduce cada secuencia de un alfabeto de entrada a una secuencia específica de un alfabeto de salida. Si **L** es el lenguaje de entrada de **M**, el

conjunto de secuencias producidas por **M** en repuesta a las secuencias de **L** es la traducción de **L** llevada a cabo por **M**.

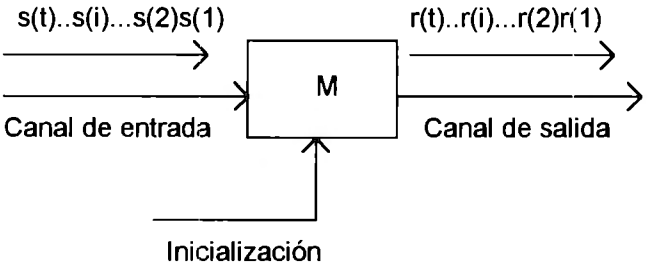


Figura 2.3. Traductor de lenguaje

Considerando cualquiera de las máquinas anteriores. Se asume que la máquina opera a instantes de tiempo $t_0, t_1, \ldots, t_i, \ldots,$. En $t = t_0$ los componentes de la máquina se encuentran inicializados a una condición de inicio conocida. En cada subsecuente momento t, **M** recibe un símbolo de entrada $s(t)$ a través de su canal de entrada y dependiendo de la máquina que se trate puede emitir un símbolo de salida $r(t)$ a través de su canal de salida. Los símbolos de entrada son elegidos de un *alfabeto de entrada* **S**:

$$s(t) \in S, \quad t = t_1, t_2, \ldots$$

Similarmente, los símbolos de salida $r(t)$ son escogidos de un *alfabeto de salida* **R**:

$$r(t) \in R, \quad t = t_1, t_2, \ldots$$

Una secuencia de símbolos de entrada presentados a la máquina es llamado *estímulo*; la secuencia resultante de símbolos de salida es llamada la respuesta de **M** al estímulo.

Se supone que **M** está constituida por un número finito de partes interconectadas, cada una de las cuales puede asumir uno de los elementos de un conjunto finito de estados. Las interconexiones son rutas a través de las cuales se transmiten las señales de una parte de la máquina a otra. Si se supone que **M** tiene n partes y que $q^{(i)}(t)$ es el estado asumido por la parte i en el momento t, entonces el *estado total* de **M** en el tiempo t es la tupla de n elementos:

$$q(t) = (q^{(1)}(t), q^{(2)}(t), \ldots, q^{(n)}(t))$$

Ya que es posible que cada parte de la máquina pueda asumir cualquiera (sólo uno) de sus estados independientemente de las otras partes, el número máximo posible de estados totales de **M** es el producto del número de estados por el número de partes. Se emplea $q(t)$ para denotar al estado (total) de **M** en el momento t , y **Q** para denotar al *conjunto finito de estados* de **M**.

Debido a que el comportamiento futuro de **M** depende claramente del presente estado de **M**, es natural averiguar el estado actual de **M** antes de presentarle cualquier entrada. A fin de que la máquina pueda tener un comportamiento determinista, es necesario que sus partes sean inicializadas a un estado fijo y conocido, antes de aplicar cualquier estímulo. El correspondiente estado total es llamado *estado inicial* de **M** y es denotado por q_0.

Una vez que se ha presentado el último símbolo de entrada a la máquina **M**, es necesario indicar sí la secuencia de entrada pertenece o no al lenguaje definido por la máquina **M**, para lo cual se define arbitrariamente un conjunto de estados de aceptación, que puede ser cualquier subconjunto de **Q**. Los correspondientes estados totales son llamados *estados finales* de **M**.

Se asume adicionalmente que la máquina cambia de estado sólo en los momentos $t_0, t_1, \ldots, t_i, \ldots$, así que su comportamiento puede ser especificado por una secuencia dada de estados $q(0), q(1), q(2), \ldots, q(t), \ldots, \quad q(t) \in Q$ que describe la condición interna de **M** en cada momento.

Cuando el símbolo de entrada $s(t+1)$ ingresa por el canal de entrada, tiene influencia en ciertas partes de la máquina, que se encuentran en una condición representada por el estado $q(t)$, estableciendo una nueva condición descrita por el estado $q(t+1)$. El nuevo estado depende únicamente del estado anterior de **M** y del símbolo de entrada $s(t+1)$. Así que, existe una función f que especifica el próximo estado de **M** en términos de su presente estado y el próximo símbolo de entrada:

$$q(t+1) = f(q(t), s(t+1)), \qquad t \geq 0$$

Esta función es llamada *función de transición* de **M**. Su dominio es el conjunto de todos los pares estado-símbolo, y su rango es un subconjunto del conjunto de estados :

$$f : Q \times S \rightarrow Q$$

El canal de salida de la máquina recibe señales de ciertas partes de **M**. El símbolo de salida generado por la llegada de un símbolo de entrada depende sólo del símbolo que se presenta a la entrada y del estado de **M** justamente antes de su llegada. Así que existe una función g que especifica el símbolo de salida producido por **M** en términos de su presente estado y de el símbolo de entrada:

$$r(t+1) = g(\, q(t), s(t+1)\,), \qquad t \geq 0$$

Esta función es llamada *función de salida* de **M**. Su dominio es el conjunto de todos los pares estado-símbolo, y su rango es el alfabeto de salida

$$g : Q \times S \rightarrow R$$

Para describir una máquina de estados finitos, es necesaria la especificación de la función de transición de estados y de la función de salida. Dos representaciones son comúnmente empleadas: La *tabla de transiciones* y el *diagrama de estados*.

La tabla de transiciones es una representación tabular de las funciones, que usa un renglón para cada estado y una columna para cada símbolo de entrada. En la posición tabular para el estado q y el símbolo s, se escribe el estado que resulta de la aplicación de la función de transición $q' = f(\, q, s\,)$ y el símbolo de salida $r = g(\, q, s\,)$ si se trata de un traductor de lenguajes.

Un diagrama de estados es un grafo dirigido en el cual cada nodo corresponde a un estado de la máquina y cada arco dirigido indica una posible transición de un estado a otro. Cada arco se encuentra etiquetado con el símbolo de entrada que produce la transición. El símbolo de salida puede etiquetar a los arcos o a los estados, dependiendo del momento en que se emite el símbolo de salida. En un diagrama de estados, el estado inicial se encuentra indicado por un arco que

no tiene nodo origen, ni símbolo de entrada. Los estados finales se encuentran indicados mediante un círculo circunscrito dentro del nodo correspondiente.

Una secuencia de estados formada por una ruta del diagrama de estados de una máquina corresponde a la aplicación de un string de símbolos de entrada y a la emisión de un string de símbolos de salida. A partir de dicha secuencia de estados es posible definir ciertos términos necesarios para la discusión del concepto de equivalencia de máquinas.

Sea **M** una máquina de estados finitos con función de transición $f: Q \times S \rightarrow Q$. Si $f(q, s) = q'$, se dice que el estado q' es el *sucesor-s* del estado q y se escribe $q \overset{s}{\rightarrow} q'$.

Si un string de símbolos de entrada $\omega = s(1)s(2)....s(t)$ lleva a **M** del estado $q = q(0)$ al estado $q' = q(t)$, estos es, si $q(0) \overset{s(1)}{\rightarrow} q(1) \overset{s(2)}{\rightarrow} \ldots \overset{s(t)}{\rightarrow} q(t)$ se dice que q' es el *sucesor-ω* del estado q y se denota $q \overset{\omega}{\Rightarrow} q'$. Bajo estas circunstancias $q(0)\,q(1)\,q(2)\ldots q(t)$ es llamada *secuencia admisible de estados* para ω.

Por medio del uso de estas definiciones se ha extendido el dominio de la función de transición de estados para incluir a todos los posibles strings de entrada, en lugar de un símbolo individual. Bajo estas condiciones se tiene

$$f: Q \times S^* \rightarrow Q$$

donde se cumple que

$$f(q, \omega) = q' \text{ si y solo si } q \overset{\omega}{\Rightarrow} q', \text{ y}$$
$$f(q, \lambda) = q \text{ para todo } q \in Q$$

Un aspecto que debe considerarse en la discusión de la teoría de autómatas es la equivalencia. En el ámbito de reconocimiento de lenguajes, dos autómatas son equivalentes si y sólo si reconocen el mismo lenguaje; en la traducción de lenguajes, dos autómatas son equivalentes si y solo si producen idénticas traducciones de cada secuencia de entrada.

En la solución del problema de equivalencia es posible además, detectar y eliminar estados redundantes sin alterar el comportamiento de la máquina y por consiguiente obtener una máquina de mínimos estados equivalente a la original.

La definición de equivalencia de máquinas se muestra a continuación:

Dos máquinas M_1 y M_2 son equivalentes si y sólo si:

1. Sus alfabetos de entrada y sus alfabetos de salida son los mismos: $S_1 = S_2$, $R_1 = R_2$.
2. Para cada estímulo, M_1 y M_2 producen idénticas respuestas. Esto es, si $s_1(t) = s_2(t), t \geq 1$, entonces $r_1(t) = r_2(t), t \geq 1$.

Si M_1 y M_2 son equivalentes, se escribe $M_1 \approx M_2$.

La definición anterior establece una relación de equivalencia en la clase de máquinas de estados finitos, puesto que muestra las siguientes propiedades:

1. *Reflexividad*: $M \approx M$. Cada máquina es equivalente a sí misma.
2. *Simetría*: Si $M_1 \approx M_2$, entonces $M_2 \approx M_1$.
3. *Transitividad*: Si $M_1 \approx M_2$ y $M_2 \approx M_3$, entonces $M_1 \approx M_3$.

La relación de equivalencia de máquinas particiona la clase de máquinas de estados finitos en colecciones de máquinas mutuamente equivalentes.

Para resolver el problema de equivalencia para máquinas de estados finitos, se debe relacionar la equivalencia de dos autómatas a su estructura interna, que se encuentra expresada por la tabla de transiciones o el diagrama de estados. Lo cual se facilita si las máquinas han sido reducidas a su forma de mínimos estados, que como se mencionó anteriormente, se consigue mediante la identificación y eliminación de estados equivalentes.

Si dos estados son equivalentes, uno de ellos es claramente redundante, dado que todas las transiciones de uno de los estados puede ser cambiadas al estado equivalente sin alterar el comportamiento de la máquina. Para identificar estados redundantes en una máquina, es necesario un método para construir clases de equivalencia de estados a partir de la tabla de transiciones o del diagrama de estados.

Intuitivamente, una máquina se encuentra en su forma más simple una vez que todos los estados redundantes y no accesibles han sido removidos. Es decir, se considera que una máquina se encuentra reducida si no contiene un par de estados equivalentes. Un estado q es *accesible* si existe una secuencia de entrada ω tal que $q_0 \overset{\omega}{\Rightarrow} q$. Una máquina de estados finitos se encuentra *conectada* si todos sus estados son accesibles a partir del estado inicial.

Es evidente que ningún estado puede ser removido de una máquina reducida y conectada sin alterar su comportamiento. Los estados inaccesibles pueden ser siempre eliminados de una máquina, ya que la máquina nunca podrá asumir dichos estados.

2.4 Máquina de Turing.

En 1936, Allan Turing publica el primer estudio de un modelo de máquina abstracta para cómputo. El concepto fue introducido con el fin de resolver un problema muy difundido, conocido como *Entscheidungsproblem*, propuesto por el matemático alemán David Hilbert en 1900. Hilbert preguntó por nada menos que un procedimiento algorítmico general para la determinación de verdad o falsedad de proposiciones matemáticas; o en su defecto, por una respuesta a la pregunta de que si tal procedimiento podía en principio existir. En 1931, Kurt Gödel publica su famoso teorema de incompletez, con el cual demuestra que tal procedimiento efectivo no podría existir. Gödel construyó una formula en el cálculo de predicados aplicada a los enteros, cuya definición afirmaba que no podría ser probada o desaprobada dentro de ese sistema lógico.

Turing, sin embargo, pretendía proporcionar una noción matemáticamente precisa para el concepto informal de procedimiento efectivo o algoritmo. Su éxito se encuentra confirmado por la continua aceptación a su tesis de que *cualquier solución algorítmica a un problema puede ser representada por el programa de instrucciones para alguna máquina de Turing*.

La máquina de Turing es un modelo matématico simple de un computador. A pesar de su simplicidad, la máquina de Turing modela las capacidades de un computador de propósito general. La máquina de Turing es estudiada tanto por

la clase de lenguajes que define (lenguajes recursivamente enumerables), como por la clase de funciones enteras que calcula (funciones recursivas parciales).

La principal característica de una máquina de Turing es su habilidad para almacenar y recuperar cantidades ilimitadas de información. El modelo básico, ilustrado en la figura 2.4, cuenta con un control finito, una cinta de entrada que se encuentra dividida en celdas, y una cabeza de cinta que trabaja con una celda a la vez. La cinta tiene una primera celda a la izquierda, pero se extiende infinitamente hacia la derecha. Cada celda de la cinta puede contener exactamente un símbolo tomado de un conjunto finito de símbolos de cinta. Inicialmente, las n celdas que se encuentran más a la izquierda, contienen la secuencia de entrada, que se construye a partir de un subconjunto de los símbolos de cinta llamado símbolos de entrada. La restante infinidad de celdas contienen al blanco, el cual es un símbolo de cinta especial que no es símbolo de entrada.

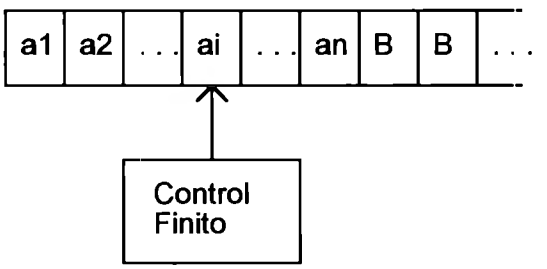


Fig. 2.4 **Máquina de Turing básica**

En un movimiento la máquina de Turing, dependiendo del símbolo leido por la cabeza de cinta y el estado que se encuentra, lleva a cabo las siguientes acciones:

- Cambia de estado,
- Imprime un símbolo en la celda de la cinta, reemplazando el símbolo que se encontraba en la celda, y
- Mueve la cabeza de cinta una celda a la derecha o a la izquierda.

Formalmente una máquina de Turing es denotada

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

Donde

Q es el conjunto finito de estados,
Γ es el conjunto finito de símbolos de cinta permitidos,
B es un símbolo de Γ, que es el blanco,
Σ es un subconjunto de Γ que no incluye a B, que es el conjuto de símbolos de entrada,
δ es la función de transición de estado, un mapeo de $Q \times \Gamma$ a $Q \times \Gamma \times \{L, R\}$,
q_0 en Q es el estado inicial,
F es un subconjunto de Q, que es el conjunto de estados finales.

Se denota a la *descripción instantánea* (ID) de la máquina de Turing M por $\alpha_1 q \alpha_2$. Donde q, es el estado actual de M; $\alpha_1 \alpha_2$ es el string en Γ^* que es el contenido de la cinta hasta el símbolo más a la derecha que no es el blanco. Finalmente, se asume que la cabeza de cinta recibirá el símbolo más a la izquierda de α_2, o si $\alpha_2 = \lambda$, la cabeza recibirá un blanco.

Se define un *movimiento* de M como sigue. Sea $X_1 X_2 \cdots X_{i-1} q X_i \cdots X_n$ una descripción instantánea. Supóngase que $\delta(q, X_i) = (p, Y, L)$, donde si $i-1=n$, entonces X_i debe ser B. Si $i=1$, entonces no existirá la siguiente descripción instantánea, ya que a la cabeza de cinta no le es permitido moverse más allá de la última celda a la izquierda de la cinta. Si $i>1$, entonces se escribe:

$$X_1 X_2 \cdots X_{i-1} q X_i \cdots X_n \vdash X_1 X_2 \cdots X_{i-2} p X_{i-1} Y X_{i+1} \cdots X_n$$

Alternativamente, si se supone que $\delta(q, X_i) = (p, Y, R)$ entonces se escribe:

$$X_1 X_2 \cdots X_{i-1} q X_i \cdots X_n \vdash X_1 X_2 \cdots X_{i-1} Y p X_{i+1} \cdots X_n$$

Si dos descripciones instantáneas se encuentran relacionadas por el símbolo $\vdash$, se dice que la segunda resulta de la primera a través de un movimiento. Si una descripción instantánea resulta de otra por medio de un número finito de movimientos, incluyendo cero movimientos, éstas se encuentran relacionadas por medio del símbolo $\overset{*}{\vdash}$.

El lenguaje aceptado por M, denotado $L(M)$, es el conjunto de todas las secuencias en Σ^* que provocan que M entre a un estado final cuando son puestas, justificadas a la izquierda, sobre la cinta de M, con M en el estado q_0, y la cabeza de la cinta colocada en la celda de más a la izquierda. Formalmente el lenguaje aceptado por $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ es:

$$\{w | w \in \Sigma^* \wedge q_0 \overset{*}{\vdash} \alpha_1 p \alpha_2 \text{ para alguna } p \in F, \text{ y } \alpha_1 \text{ y } \alpha_2 \in \Gamma^* \}.$$

Dada una máquina de Turing que reconoce al lenguaje L, se asume sin pérdida de generalidad que la máquina de Turing se detiene, es decir, no puede realizar el movimiento siguiente, una vez que la secuencia de entrada ha sido aceptada. Sin embargo, para secuencias que no se encuentran dentro del lenguaje L, es posible que la máquina de Turing nunca se detenga.

2.5 Lenguajes y Funciones Computables.

Un lenguaje que es aceptado por una máquina de Turing es llamado *recursivamente enumerable*. El término *enumerable* se deriva del hecho de que son precisamente los lenguajes cuyos strings pueden ser enumerados por una máquina de Turing. *Recursivamente* es un término matemático cuyo significado es similar a lo que un científico de computación llamaría recursión. La clase de lenguajes recursivamente enumerables es muy amplia e incluye a los lenguajes libres de contexto.

La clase de lenguajes recursivamente enumerables incluye algunos lenguajes para los cuales no es posible determinar mecánicamente la membresía. Si $L(M)$ es tal lenguaje, entonces cualquier máquina de Turing que reconoce a $L(M)$ debe fallar en detenerse sobre alguna secuencia de entrada que no se encuentra en $L(M)$. Si w está en $L(M)$, M eventualmente se detendrá con la secuencia w. Sin embargo, mientras que M continúe corriendo sobre alguna secuencia de entrada, nunca se podrá decir, ya sea que, M eventualmente se detendrá si se le permite seguir corriendo por algún tiempo suficientemente grande, o que M correrá para siempre.

Es conveniente señalar una subclase de lenguajes recursivamente enumerables llamada *conjuntos recursivos*, la está conformada por los lenguajes que son

aceptados por lo menos por una máquina de Turing que se detiene con todas las secuencias de entrada en $L(M)$.

Además de ser un aceptador de lenguajes, la máquina de Turing puede ser considerada como un traductor de lenguajes que calcula funciones de números enteros a números enteros. El enfoque tradicional es representar los enteros en *unario*; el entero $i \geq 0$ es representado por el medio del string 0^i. Si una función tiene k argumentos, $i_1, i_2, \ldots, i_k$, entonces estos enteros son puestos inicialmente en la cinta separados por 1's, como $0^{i_1}10^{i_2}1\cdots10^{i_k}$.

Si la máquina de Turing se detiene (ya sea que se encuentre en un estado final o no) con una cinta que contiene a 0^m para alguna m, entonces se dice que $f(i_1, i_2, \ldots, i_k) = m$, donde f es la función de k argumentos calculada por esta máquina de Turing. Nótese que una máquina de Turing puede computar una función de un argumento, una función diferente de dos argumentos, y así sucesivamente. También debe notarse que si una máquina de Turing M computa una función f de k argumentos , entonces f no necesariamente tendrá un valor para todas las diferentes tuplas de k enteros $i_1, i_2, \ldots, i_k$.

Si $f(i_1, \ldots, i_k)$ se encuentra definida para toda $i_1, \ldots, i_k$, entonces se dice que f es una *función recursiva total*. Una función $f(i_1, \ldots, i_k)$ computada por una máquina de Turing es llamada *función recursiva parcial*. En algún sentido, las funciones recursivas parciales son análogas a los lenguajes recusivamente enumerables, ya que son computadas por máquinas de Turing que pueden o no detenerse con una secuencia de entrada dada. Las funciones recursivas totales corresponden a los lenguajes recursivos, ya que son computados por máquinas de Turing que siempre se detienen. Todas las funciones aritméticas de los enteros, tales como la multiplicación, $n!$, $\log_2 n$, y 2^{2^n} son funciones recursivas totales.

Una de las razones por las cuales el modelo expuesto con anterioridad ha llegado a ser reconocido como modelo general, es que ha demostrado ser equivalente a muchas versiones ampliadas (máquina de Turing con cinta infinita en ambos lados, multicinta, multicabeza, multidimensional, etc.) que en apariencia incrementan el poder de cómputo del modelo general. De igual manera existen modelos restringidos (máquina de Turing multistack, máquina

contadora, etc.) que aparentemente cuentan con menor capacidad de cómputo que el modelo general, sin embargo, han demostrado ser igualmente poderosos.

Capítulo 3

Estrategias Evolutivas y Algoritmos Genéticos

3.1 Introducción.

Las *estrategias evolutivas* y los *algoritmos genéticos* son procedimientos que imitan los principios de la evolución natural como método para resolver problemas de optimización. Estas dos metodologías surgieron más o menos al mismo tiempo y tienen muchas características en común, la más importante es que exhiben un mejor desempeño en la solución de cierto dominio de problemas, que los métodos tradicionales basados en el cálculo.

La idea de utilizar los procesos de la evolución biológica como reglas que guían los procedimientos de búsqueda para lograr soluciones óptimas surgieron independientemente en Alemania y en Estados Unidos hace más de dos décadas. Ambas metodologías imitan operadores y paradigmas de procesamiento de información, como la reproducción y selección de poblaciones naturales, con el fin de construir algoritmos de optimización robustos y eficientes. Estos algoritmos pueden ser vistos como la representación algorítmica o matemática de la teoría de la evolución de Darwin [Holl75].

Hacia el final de la década de los sesenta, en Alemania se desarrollaron los primeros trabajos en procedimientos basados en los principios de evolución. Rechenberg en 1973 con su publicación *Evolutionsstrategie* [Rech73], presenta la metodología llamada *estrategias evolutivas*, que fue empleada ampliamente en la solución de problemas de optimización.

Por otro lado, en Estados Unidos, Holland en 1975, con su libro *Adaptation in Natural and Artificial Systems* [Holl75], establece las bases de los algoritmos genéticos como parte fundamental de los sistemas adaptativos. Este trabajo hace mención de la aplicabilidad de los algoritmos genéticos a los problemas de optimización de paramétros, lo que fue experimentado inicialmente por De Jong. Actualmente los algoritmos genéticos son empleados en la solución de una gran variedad de problemas.

3.2 Problemas de Optimización.

Los problemas de optimización pertenecen a una clase de problemas cuya solución puede obtenerse a través de un algoritmo de búsqueda, en el que se trata de encontrar un elemento particular, dentro de un espacio de búsqueda cuyos estados se definen por una colección de parámetros, cada uno de los cuales toma valores dentro de un dominio finito.

En la mayoría de los casos los problemas de optimización admiten soluciones aproximadas, debido a ésto, cualquier procedimiento sistemático que garantice un porcentaje de errores inferior al cincuenta por ciento es una solución aceptable cuando no otra alternativa, que no sea la elección estocástica. Así que, aunque es deseable obtener la solución óptima, no es absolutamente necesaria para tener un sistema que funciona adecuadamente.

Los problemas de optimización pueden ser formulados matemáticamente mediante la consideración de la noción de distancia o criterio de mérito. En su planteamiento más general, un problema de optimización se reduce a encontrar el valor máximo (o mínimo) de una función $f: D \rightarrow \Re$, llamada función objetivo, o función de mérito. En los casos más sencillos D es algún subconjunto del espacio real de n dimensiones, no obstante, puede ser cualquier conjunto de entidades [Chav93].

Cuando $D \subset \Re^n$ y en la función f las derivadas parciales de primer orden existen y son continuas, la determinación del máximo (o mínimo) de f se reduce al análisis de los valores de la función en los puntos críticos, donde se anulan las derivadas:

$$\frac{\partial f}{\partial x_i} = 0; (i = 1, \cdots, n),$$

La desventaja principal del método de descenso a lo largo de las líneas del gradiente, es que emplea únicamente información local para guiar la búsqueda de valores óptimos en el espacio de parámetros, por lo que es fácil localizar máximos (o mínimos) locales cuando la topología de la función es complicada, después de lo cual la optimización se interrumpe. Otra desventaja del método es que requiere de información adicional al valor de la función objetivo, como sus derivadas, que pueden no estar definidas en muchos casos prácticos [Chav93].

Es posible emplear un procedimiento exhaustivo de enumeración de las distintas posibilidades para la optimización de funciones en espacios finitos de búsqueda o espacios infinitos discretizados. En este caso, el algoritmo de búsqueda explora los valores de la función objetivo en todos y cada uno de los puntos del espacio, uno a la vez. Sin embargo, a pesar de la sencillez del procedimiento, en la mayoría de los casos el número de posibilidades será tan grande que este método carece de valor práctico.

Los procedimientos estocásticos de búsqueda han alcanzado cierta popularidad a medida que se han demostrado las deficiencias de los procedimientos basados en cálculo y de los procedimientos enumerativos. Sin embargo, la caminata al azar, a la larga, se espera que tenga un desempeño no mejor a los procedimientos enumerativos. De esta manera, los procedimientos estocásticos son generalmente descartados debido a requerimientos de eficiencia en la solución del problema [Gold89].

3.3 Estratégias Evolutivas.

Las estrategias evolutivas son algoritmos que imitan los principios de la evolución natural como método para la solución de problemas de optimización de parámetros. Las estrategias evolutivas fueron introducidas por Rechenberg en la Universidad de Berlín [Rech73], y desarrolladas ampliamente por Schwefel [Schw75].

Rechenberg [Rech73] propuso utilizar la metodología de estrategias evolutivas en la resolución de problemas de optimización de una función objetivo $f:\Re^n \to \Re$, sujeta a las condiciones de restricción $\varphi_i(x) \le 0,(i=1,\cdots,m)$, sin emplear información sobre sus derivadas parciales. En el caso más simple, se utiliza una población de dos elementos (vectores en el espacio n-dimensional), $\vec{x}^{(1)} = \left(x_1^{(1)},\cdots,x_n^{(1)}\right)$ llamado padre y $\vec{x}^{(2)} = \left(x_1^{(2)},\cdots,x_n^{(2)}\right) = \left(x_1^{(1)}+\delta_1,\cdots,x_n^{(1)}+\delta_n\right)$, llamado hijo (sucesor), cuyas componentes se obtienen al añadir fluctuaciones δ_i normalmente distribuidas , a cada una de las componentes correspondientes de $\vec{x}^{(1)}$ En el caso en que se trata de obtener el valor mínimo de la función objetivo, entonces, si:

$$f\left(x^{(2)}\right) < f\left(x^{(1)}\right) \& \left(\varphi_i\left(x^{(2)}\right) \le 0 \forall i\right)$$

$x^{(2)}$ reemplaza a $x^{(1)}$ en el papel de padre para la siguiente generación, de esta manera, se puede decir que sobrevive el mejor. Rechenberg [Rech73] nombró a esta estrategia $(1+1)-ES$. Formalmente, el procedimiento $(1+1)-ES$ se describe a partir de una tupla de 6 elementos:

$$(1+1)-ES = (P,m,S,f,\varphi_i,t)$$

Donde:

$P = \left(\vec{x},\sigma^2\right) \in \Re^n \times \Re = \Re^{n+1}$	Población
$m:\Re^{n+1} \to \Re^{n+1}$	Operador de Mutación
$S:\Re^{n+1} \times \Re^{n+1} \to \Re^{n+1}$	Operador de Selección
$f:\Re^n \to \Re^n$	Función objetivo
$\varphi_i:\Re^n \to \Re,(i=1,\cdots,m)$	Condiciones de restricción
$t:\Re^n \times \Re^n \to \{0,1\}$	Criterio de terminación

Lo población inicial consiste de un solo padre, cuyas componentes se inicializan aleatoriamente observando que se cumplan las condiciones de restricción $\forall i: \varphi_i(\vec{x}) \leq 0, (i = 1, \cdots, m)$.

El padre produce, mediante el operador de mutación un hijo único $P' = \left(\left(\vec{x}^{(1)}, \sigma^2\right), \left(\vec{x}^{(2)}, \sigma^2\right)\right)$, donde $\vec{x}^{(1)} = \vec{x}$ y $\vec{x}^{(2)} = \vec{x} + \vec{N}(\vec{0}, \sigma)$ y $\vec{N}(\vec{0}, \sigma)$ es un vector aleatorio con componentes independientes, idéntica y normalmente distribuidas con valor medio cero y varianza σ^2.

Una vez que se tiene a ambos elementos de la población se procede a aplicar el operador de selección a P', para obtener el nuevo valor de P:

$$S\left(\left(\vec{x}^{(1)}, \sigma^2\right), \left(\vec{x}^{(2)}, \sigma^2\right)\right) = \begin{cases} \left(\vec{x}^{(2)}, \sigma^2\right) \Leftrightarrow f\left(\vec{x}^{(2)}\right) \leq f\left(\vec{x}^{(1)}\right) \& \left(\forall i: \varphi_i\left(\vec{x}^{(2)}\right) \leq 0\right) \\ else \left(\vec{x}^{(1)}, \sigma^2\right) \end{cases}$$

Si al aplicar este operador se cumple la condición de terminación presentar $\vec{x}$ y terminar, si no, el elemento seleccionado como padre para la siguiente generación produce un nuevo hijo, convirtiendo a esta metodología en un procedimiento iterativo.

Para introducir el concepto de población al algoritmo, Rechenberg [Rech73] propuso la estrategia evolutiva $(\mu+1)-ES$ que es similar a la anterior, salvo que la población inicial consiste de μ $(\mu > 1)$ elementos. En este caso, es posible imitar la reproducción sexual. Una $(\mu+1)-ES$ puede se formalizada como:

$$(\mu+1)-ES = (P, R, m, S, f, \varphi_i, t)$$

Donde:

$P = \left(\left(\vec{x}^{(1)}, \sigma_1^2\right), \cdots, \left(\vec{x}^{(\mu)}, \sigma_\mu^2\right)\right), \left(\vec{x}^{(i)}, \sigma_i^2\right) \in \Re^{n+1}$ Población

$R: \left(\Re^{n+1}\right)^\mu \rightarrow \left(\Re^{n+1}\right)^{\mu+1}$ Operador de Recombinación

$m: \Re^{n+1} \rightarrow \Re^{n+1}$	Operador de Mutación
$S: \Re^{n+1} \times \Re^{n+1} \rightarrow \Re^{n+1}$	Operador de Selección
$f: \Re^n \rightarrow \Re^n$	Función objetivo
$\varphi_i: \Re^n \rightarrow \Re, (i = 1, \cdots, m)$	Condiciones de restricción
$t: \Re^n \times \Re^n \rightarrow \{0,1\}$	Criterio de terminación

En este caso, como primer paso se inicializa aleatoriamente la población:

$$P = \left(\left(\vec{x}^{(1)}, \sigma_1^2\right), \cdots, \left(\vec{x}^{(\mu)}, \sigma_\mu^2\right)\right), \left(\vec{x}^{(i)}, \sigma_i^2\right) \in \Re^{n+1}$$

de tal manera que se cumplan las condiciones de restricción:

$$\varphi_i: \Re^n \rightarrow \Re, (i = 1, \cdots, m).$$

Posteriormente se aplica el operador de recombinación para obtener la población intermedia:

$$P' = \left(\left(\vec{x}^{(1)}, \sigma_1^2\right), \cdots, \left(\vec{x}^{(\mu+1)}, \sigma_{\mu+1}^2\right)\right).$$

En el siguiente paso se emplea el operador de mutación para obtener una nueva población intermedia:

$$P'' = \left(\left(\vec{x}^{(1)}, \sigma_1^2\right), \cdots, m\left(\vec{x}^{(\mu+1)}, \sigma_{\mu+1}^2\right)\right).$$

Una vez obtenida la población P'', se aplica el operador de selección. Si se cumple la condición de terminación se suspende el procedimiento, si no, se procede a aplicar repetidamente el operador de recombinación y los operadores subsecuentes.

La principal diferencia entre las estrategias evolutivas $(\mu+1)-ES$ y $(1+1)-ES$ radica en el número de elementos de la población y en los operadores de recombinación y selección. La recombinación es un modelo de la reproducción sexual. El operador de recombinación escoge internamente dos elementos de la población para reproducción $\left(\bar{x}^{(a)}, \sigma_a^2\right)$ y $\left(\bar{x}^{(b)}, \sigma_b^2\right)$ ya sea completamente al azar, o mediante la consideración de una densidad de probabilidad que considere valores de la función objetivo.

La acción del operador de mutación nos permite obtener entonces la población P'', sobre la cual actúa el operador de selección, que simplemente descarta el peor de los elementos de la población, ya sea por no cumplir las restricciones o por observar un criterio de mérito bajo.

Un sinnúmero de extensiones a los procedimientos $(\mu+1)-ES$ y $(1+1)-ES$ han sido llevadas a cabo por Schwefel [Schw75], tal es el caso de los procedimientos $(\mu+\lambda)-ES$ y $(\mu,\lambda)-ES$ los cuales favorecen la utilización de computadoras paralelas y la adaptación propia de parámetros estratégicos como, por ejemplo, la desviación estándar de las mutaciones.

3.4 Algoritmos Genéticos

Los algoritmos genéticos surgen en Estados Unidos simultáneamente con las estrategias evolutivas. J. H. Holland, profesor de la Universidad de Michigan en su libro *Adaptation in Natural and Artificial Systems* [Holl75] describe como los procesos de evolución de la naturaleza pueden ser aplicados a sistemas artificiales. En particular los algoritmos genéticos son procedimientos matemáticos altamente paralelos que transforman una población de elementos del espacio n-dimensional en una nueva población utilizando los principios de la genética y la selección natural.

La principal ventaja de los algoritmos genéticos es que al igual que las estrategias evolutivas, requieren únicamente de información relativa a los valores de la función objetivo para llevar a cabo la optimización. De la misma manera utilizan procedimientos estocásticos para guiar la búsqueda en el espacio de parámetros, y llevan a cabo la búsqueda simultáneamente en diferentes puntos.

Los algoritmos genéticos consideran una población inicializada aleatoriamente de cadenas binarias de longitud N que corresponden a la representación de los parámetros de optimización. Cada uno de los elementos de la población representa, un punto en el espacio real n-dimensional:

$$(b_1,\cdots,b_N)\rightarrow(x_1,\cdots,x_n)$$

lo que permite asociar un número real a cada cadena binaria de longitud N:

$$(b_1,\cdots,b_N)\rightarrow(x_1,\cdots,x_n)\rightarrow f(x_1,\cdots,x_n)$$

que es el valor de la función objetivo en el punto correspondiente, con lo cual se ha proyectado el problema de optimización original en un problema de optimización de números enteros.

En el caso más simple un algoritmo genético está compuesto por tres operadores [Gold89]:

1. Operador de reproducción
2. Operador de cruzamiento
3. Operador de mutación

La reproducción es el proceso en el cual las cadenas binarias son copiadas de acuerdo al valor de la función objetivo, f (los biólogos llaman a ésta la función de fitness). Copiar las cadenas de acuerdo al valor del fitness significa que las cadenas con los valores más altos tienen una mayor probabilidad de contribuir con uno o más hijos en la próxima generación. Este operador es una versión artificial de la selección natural, el principio de Darwin de sobrevivencia del mejor.

El operador de reproducción puede ser implementado de diferentes maneras, la más sencilla, es crear una ruleta en donde, a cada string de la población corresponde un sector de la ruleta de tamaño proporcional a su valor de fitness. Cada vez que se requiere un hijo, un simple giro de la ruleta genera un posible candidato para reproducción. De esta manera se obtiene una población intermedia.

Si se trata de obtener el valor máximo de una función positiva, esto se puede implementar, por ejemplo, mediante la densidad de probabilidad:

$$p_i = \frac{f\left(\vec{x}^{(i)}\right)}{\sum_{j=1}^{n} f\left(\vec{x}^{(j)}\right)}$$

Con función de distribución $P_i = \sum_{j=1}^{i} p_i$

Para muestrear el índice aleatorio i con esta densidad de probabilidad se puede emplear:

$$m = 1;$$
$$z = \aleph;$$
$$if\,(P_j < z) return\, m$$
$$else\; m = m + 1;$$

Donde $\aleph$ es una variable aleatoria uniformemente distribuida en el intervalo $[0,1]$, m es el índice del *m-esímo* elemento de la población.

Después de la reproducción, un simple cruzamiento es efectuado sobre los miembros de la población intermedia. Esto se lleva a cabo mediante la selección aleatoria de parejas de strings. Una posición entera k a lo largo del string es seleccionada aleatoriamente entre 1 y la longitud del string menos uno $[1, l-1]$. Dos nuevos strings son creados al intercambiar todos los caracteres entre las posiciones $k+1$ y l inclusive. Esto es:

$$R(b_1 \cdots b_N, b_1' \cdots b_N') = \begin{cases} \aleph < p_c \,?(b_1 \cdots b_a b_{a+1}' \cdots b_N', b_1' \cdots b_k' b_{k+1} \cdots b_N) \\ \aleph > p_c \,?(b_1 \cdots b_N, b_1' \cdots b_N') \end{cases}$$

Donde $\aleph$ es una variable aleatoria uniformemente distribuida en el intervalo $[0,1]$, p_c es la probabilidad de cruzamiento en la práctica es del orden de $p_c \approx .6$.

En el algoritmo genético simple, la mutación es la alteración aleatoria ocasional (con pequeña probabilidad) del valor de una posición del string. Si la codificación es binaria, esto simplemente significa cambiar un 0 por un 1 o viceversa. Esto es:

$$m(b_1,\cdots,b_N)=\left(flip(b_1),\cdots,flip(b_N)\right)$$

$$flip(b)=\begin{cases}\aleph < p_m : !b \\ \aleph > p_m : b\end{cases}$$

Donde $\aleph$ es una variable aleatoria uniformemente distribuida en el intervalo $[0,1]$, p_m es la llamada probabilidad de mutación. La probabilidad de mutación que se usa en la práctica es del orden de $p_m \approx 0.01$. Por sí misma la mutación es una caminata al azar a través del espacio de strings que evita la localización de mínimos locales.

El algoritmo procede de la siguiente forma [Chav93]:

Paso 1. Una población P de tamaño n compuesta de cadenas binarias de longitud N es inicializada aleatoriamente: $P=\left(b_1^{(1)}\cdots b_N^{(1)},\cdots,b_1^{(m)}\cdots b_N^{(m)}\right)$. El valor de estas soluciones aleatorias es evaluado con la función objetivo $b_1^{(1)}\cdots b_N^{(1)}\rightarrow f^{(1)},\cdots,b_1^{(m)}\cdots b_N^{(m)}\rightarrow f^{(n)}$

Paso 2. Una población intermedia $P'=\left(b_1^{(1)}\cdots b_N^{(1)},\cdots,b_1^{(m')}\cdots b_N^{(m')}\right)$, se obtiene al seleccionar aleatoriamente elementos de la población con una densidad de probabilidad que refleje las diferencias entre los valores de la función objetivo.

Paso 3. Pares de elementos en la población P' son seleccionados aleatoriamente para reproducción, tantos como sean necesarios para sustituir la población P, tomando en cuenta que cada uno de estos pares tendrá dos descendientes en la siguiente generación.

Paso 4. El operador de cruzamiento es aplicado a cada uno de los pares anteriormente seleccionados para obtener sus dos descendientes.

Paso 5. El operador de mutación es aplicado a cada uno de los descendientes obtenidos en el punto anterior.

Paso 6. Si se cumple la condición de terminación (se ha obtenido el valor óptimo o se ha alcanzado el número predeterminado de generaciones), entonces terminar, si no regresar al paso 2.

3.5 Teorema de los Esquemas.

El teorema de los esquemas fue construido por Holland para demostrar el procesamiento de información y el poder de convergencia de los algoritmos genéticos [Holl75]. El teorema de los esquemas es principalmente demostrativo, respecto al efecto que tienen los operadores genéticos (reproducción, cruzamiento y mutación) en la adaptación y optimización.

El aspecto más importante de un algoritmo de optimización, especialmente cuando se trata de algoritmos heurísticos en dominios complejos, es que debe existir un balance entre la exploración de nuevas regiones en el espacio de soluciones y la explotación de buenas regiones que ya han sido descubiertas por el algoritmo. El teorema de los esquemas explica como este balance es llevado a cabo implícitamente por los algoritmos genéticos, a través del uso de los operadores genéticos [Gold89].

Se introduce la noción de un esquema H con la adición del símbolo $*$, que se utiliza para indicar determinadas posiciones de la tupla correspondiente a los elementos de la población, en donde el símbolo del alfabeto (cualquiera que sea) que aparece codificando dicha posición carece de importancia. Un hiperplano de orden k de $\Re^n$ es definido como un subespacio $(l-k)$-dimensional de $\Re^n$, especificado al fijar exactamente k de las l posiciones de la tupla y definiendo las posiciones restantes como carentes de importancia con el símbolo $*$. Un miembro de este hiperplano es conocido como *esquema*, y un esquema define un hipercubo que contiene todas las tuplas que coinciden al esquema en sus k posiciones especificadas.

Un esquema representa un subespacio del espacio de soluciones. Dado que se desea dedicar mayores esfuerzos a la búsqueda en aquellas regiones donde el

espacio de solución parece prometedor, los esquemas son una herramienta adecuada para designar tales regiones.

Por ejemplo, si el valor promedio de la función objetivo de las tuplas en P que comienzan con 0 (Aquellas tuplas que son instanciadas por el esquema $0{**}\cdots{*}$) es más alto que aquellas tuplas que comienzan con 1, entonces, en promedio, las tuplas que comienzan con 0 serán replicadas más frecuentemente que aquellas que comienzan con 1. Esto produce un incremento en la frecuencia de las tuplas que comienzan con 0 y de la frecuencia del esquema $0{**}\cdots{*}$.

Como resultado de la replicación proporcional y la uniforme eliminación de tuplas, se sigue la implícita replicación de esquemas. Si se denota al número de tuplas en P en el tiempo t que pertenecen al esquema H como $M(H,t)$, entonces el número de instancias del esquema H en la siguiente generación, $M(H,t+1)$, es:

$$M(H,t+1) = M(H,t)\frac{f_h(t)}{\bar{f}(t)} \qquad \text{donde} \qquad f_H(t) = \frac{\sum_{tuplas \in H} f(g(x))}{\sum_{tuplas \in H}}$$

y $f(g(x))$ es el valor de la función objetivo para el elemento x de la población y

$\sum_{tuplas \in H}$ es la suma de las tuplas que pertenecen al esquema H en al tiempo t.

es el valor promedio del esquema calculado a partir de la población P y $\bar{f}(t)$ es el valor promedio de la función para todas las tuplas (esquemas) en P. Para obtener una medida cualitativa de la razón de crecimiento, se asume que un esquema particular, permanece por encima del promedio por una cantidad fija $c\bar{f}(t)$ (siendo c una constante) para un número de ciclos t. Esto nos lleva a la consideración del siguiente crecimiento exponencial:

$$M(H,t) = M(H,0)\left[\frac{\bar{f}(t) + c\bar{f}(t)}{\bar{f}(t)}\right]^t = M(H,0)(1+c)^t$$

Lo mismo se cumple para todos los esquemas, ya sea que tengan un valor por encima o por debajo del promedio.

El operador de reproducción del algoritmo no introduce soluciones nuevas, ya que los puntos (tuplas en P) que se replican son los ya existentes. Los nuevos puntos de búsqueda son generados por los operadores de mutación y cruzamiento. Está claro que la creación de nuevas tuplas puede interrumpir el crecimiento exponencial discutido anteriormente. Con el fin de estimar el efecto ocasionado por los dos operadores en el crecimiento de los esquemas, se introducen dos conceptos: la longitud definida de un esquema $\delta(H)$, y el orden de un esquema $o(H)$.

La longitud definida de un esquema se define como la distancia entre las posiciones más lejanas del la tupla que se encuentran fijas. Consecuentemente, la probabilidad de que el operador de cruzamiento destruya un esquema dado H es la probabilidad de que el punto de cruzamiento se encuentre dentro de algunas de las posiciones dentro de la región específica del esquema y es igual a $P_c \delta(H) / (l-1)$.

El orden de un esquema denota el número de posiciones fijas en el esquema (las k posiciones). La probabilidad de destruir un esquema debido al operador de mutación es $o(H)^{P_m} \approx P_m o(H)$; $P_m << 1$. Esto nos lleva a la consideración de una aproximación de crecimiento de los esquemas, dada la utilización de los operadores de reproducción, cruzamiento y mutación, designada por Holland como *Teorema de los esquemas*:

$$M(H,t+1) = M(H,t)\frac{f_H(t)}{\bar{f}(t)}\left[1 - P_c\frac{\delta(H)}{l-1} - P_m o(H)\right]$$

Otros avances en el campo de los algoritmos genéticos se han llevado a cabo en los últimos años. Estudios comparativos del efecto de diferentes operadores de cruzamiento, formato de representación y tamaño de población indican que los tres se encuentran fuertemente ligados. Una gran cantidad de esfuerzos en investigación son dedicados a profundizar en el entendimiento de los aspectos

de representación, adaptación y procesamiento de información asociados a los algoritmos genéticos y sus aplicaciones.

3.6 Programación Genética.

Existe una variedad de problemas en el campo de Aprendizaje de Máquina (Machine Learning), Inteligencia Artificial, y Procesamiento Simbólico, que pueden ser formulados como el requerimiento de encontrar un programa de computadora que produce una salida deseada para entradas particulares. Visto de esta manera, el proceso de resolver estos problemas es equivalente a llevar a cabo una búsqueda en el espacio de programas de computadora, con el fin de encontrar el mejor. El paradigma de programación genética desarrollado recientemente por John Koza en la Universidad de Stanford [Koza92], provee una metodología para llevar a cabo la búsqueda en el espacio de programas de computadora y que permite resolver una gran variedad de problemas de diferentes campos.

En el paradigma de programación genética, poblaciones de programas de computadora evolucionan genéticamente usando el principio de Darwin de supervivencia del mejor y un operador genético de cruzamiento (recombinación sexual). Las estructuras sujetas a adaptación en el paradigma de programación genética son programas jerárquicos de computadora con tamaño y forma que varía dinámicamente.

El procedimiento comienza con una población inicial de programas aleatoriamente generados compuestos de funciones apropiadas para el dominio del problema. Las funciones pueden ser operaciones aritméticas comunes y corrientes, operaciones de programación, funciones matemáticas, funciones lógicas, y funciones de dominio específico.

Cada programa de computadora es evaluado en términos de desempeño en el medio ambiente del problema. De esta manera, algunos elementos de la población son mejores que otros. El principio de Darwin de reproducción y supervivencia del mejor, así como un operador genético de recombinación son usados para crear una nueva población de programas. En particular, un proceso genético de reproducción sexual entre dos programas padres es usado para crear programas hijos. Los programas padres son seleccionados de acuerdo a un criterio de mérito. Los programas hijos resultantes están

compuestos de subexpresiones (subárboles, subprogramas, subrutinas) de sus padres. De este modo la población anterior es sustituida por la población de programas hijos, y el procedimiento se repite nuevamente.

El paradigma de programación genética ha sido empleado con éxito en la solución de problemas de regresión simbólica, planeación, control óptimo, comportamiento emergente, etc.

En conclusión, los algoritmos genéticos constituyen un procedimiento sistemático robusto y eficiente en la solución de problemas de optimización, por lo que se ha elegido como mecanismo de búsqueda en la implementación del modelo de generación automática de máquinas de Turing.

Capítulo 4

Modelo de Generación Automática de Máquinas de Turing

4.1 Modelo Aumentado de Máquina de Turing.

El modelo de generación automática de máquinas de Turing que se ha desarrollado en el presente trabajo de investigación considera la búsqueda de una máquina de Turing que presente un mejor desempeño, ya sea en la definición de un lenguaje que describe a un conjunto de secuencias, o en el cálculo de funciones recursivas. La búsqueda exhaustiva de un dispositivo de esta naturaleza es, si el número de estados y/o alfabeto de cinta es grande, muy costosa computacionalmente. Un algoritmo genético se ha desarrollado en un sistema computacional para llevar a cabo la búsqueda de la mejor (o casi mejor) máquina de Turing.

El modelo de máquina originalmente propuesto por Allan Turing, presenta una cinta de tamaño infinito por la derecha, lo que no es práctico considerar como parte del modelo dado los recursos limitados de almacenamiento de la computadora. Además, se sabe por el *problema de Halting* [Hopc79] para máquinas de Turing, que el problema de determinar si una máquina se detendrá con una entrada determinada es indecidible. Por lo que, para hacer posible la simulación del modelo, se procede a desarrollar un modelo aumentado de máquina de Turing.

Con el modelo aumentado de máquina de Turing que se ha desarrollado se imponen ciertas restricciones al modelo original con el fin de hacer viable

(computacionalmente) la búsqueda mediante el uso de algoritmos genéticos. Denotaremos este modelo por medio de la tupla de nueve elementos:

$$M = \{Q, \Sigma, \Gamma, \delta, q_0, B, q_f, \Omega, \Pi\}$$

Donde:

Q es un conjunto finito de estados

Γ es un conjunto finito de símbolos de cinta permitidos

B es un símbolo de Γ, que es blanco. Particularmente Γ_{n-1}, donde $n = |\Gamma|$.

Σ es un subconjunto de Γ sin incluir a B, es el conjunto de símbolos de entrada

δ es la función de transición, un mapeo de $Q \times \Gamma$ a $Q \times \Gamma \times \{L, R\}$.

q_0 en Q es el estado inicial.

q_f en Q es el estado final. Particularmente q_{n-1}, donde $n = |Q|$.

Ω es un entero positivo que representa el tamaño de la cinta de entrada.

Π es un entero positivo que representa el número máximo de movimientos permitidos.

Tomando en consideración el modelo aumentado de máquina de Turing, una máquina particular recibe como entrada una cinta finita de tamaño Ω y realiza como máximo Π transiciones. Se propone que en el caso en de una transición de la máquina de Turing indique realizar un movimiento hacia la derecha de la cabeza, dado que ésta se encuentra de la posición Ω, se interrumpe la ejecución y se considere a la cinta de entrada como rechazada por la máquina de Turing. Lo mismo ocurre cuando se ha alcanzado Π número de transiciones en la ejecución de la máquina.

Se ha determinado que las instancias del modelo aumentado de máquina de Turing deben exhibir un comportamiento determinista, ya que si se utiliza el enfoque no determinista, la complejidad computacional (temporal principalmente) de la simulación se incrementa considerablemente. Para agilizar la convergencia del algoritmo, se sugiere en cambio, considerar una población relativamente grande (más de 256 elementos) y mecanismos de regulación de copias en el algoritmo genético (escalamiento del fitness) ya que de esta manera se asegura la diversidad en los elementos de la población (véase capítulo 3).

4.2 Implementación del Algoritmo Genético.

Para llevar a cabo la implementación de un algoritmo genético es necesario determinar la representación en cadenas de longitud fija de los elementos de la población, una función objetivo para evaluar a la población durante la búsqueda, un conjunto de operadores genéticos para la generación de poblaciones subsecuentes, y una asignación de probabilidades para el control de los operadores genéticos. Se propone que la construcción de un algoritmo genético para cualquier problema sea separado en cuatro distintas actividades:

1. Selección del esquema de representación de la población.
2. Diseño de los operadores genéticos.
3. Determinación de la función objetivo.
4. Selección de probabilidades para la aplicación de operadores genéticos.

4.2.1 Selección del Esquema de Representación.

Se presume que la manera más sencilla de llevar a cabo la representación de los elementos de la población (máquinas de Turing) se presenta al considerar la codificación de la tabla de transiciones de la máquina de Turing. A continuación se muestra el esquema de representación que se propone.

En la función de transición δ, para cada par estado-símbolo (q,a) en $Q \times \Gamma$, existe una tupla de tres elementos (q',a',m) los cuales representan respectivamente, el siguiente estado que asume la máquina, el símbolo que se imprime en la cinta y el movimiento (izquierda o derecha) de la cabeza de

lectura escritura. Por lo que sugiere representar a la secuencia de caracteres que será manipulada por el algoritmo genético mediante un vector con las siguientes componentes:

$$q_{0,0}\, a_{0,0}\, m_{0,0}\; q_{0,1}\, a_{0,1}\, m_{0,1} \ldots q_{1,0}\, a_{1,0}\, m_{1,0}\; q_{1,1}\, a_{1,1}\, m_{1,1} \ldots\; q_{m,n}\, a_{m,n}\, m_{m,n}$$

cuyos índices de cada componente representan respectivamente el índice en el vector de estados y el índice en el vector de símbolos que tienen las componentes del par (q, a) que genera la transición. Además se tiene que $m = |Q| - 1$, $n = |\Gamma| - 1$.

Considerando la representación anterior, es posible la definición de un esquema de direccionamiento que permite accesar la tupla $(q'_{i,j}, a'_{i,j}, m_{i,j})$ que corresponde al par (q_i, a_j), de la siguiente manera:

$$index_{(q_i, a_j)} = 3\left(i\,|\Gamma| + j\right)$$

Donde $index_{(q_i, a_j)}$ representa la posición en la cadena en donde se encuentra el primer elemento de la tupla $(q'_{i,j}, a'_{i,j}, m_{i,j})$ del par (q_i, a_j), i es la posición del estado q en el vector formado con los elementos de Q y j es la posición del símbolo a en el vector correspondiente a los elementos de Γ.

A fin de poder llevar a cabo la implementación computacional se requiere de las siguientes estructuras de datos:

Un vector *population* en cual cada uno de los componentes es un vector de la forma:

$$q_{0,0}\, a_{0,0}\, m_{0,0}\; q_{0,1}\, a_{0,1}\, m_{0,1} \ldots q_{1,0}\, a_{1,0}\, m_{1,0}\; q_{1,1}\, a_{1,1}\, m_{1,1} \ldots q_{j,k}\, a_{j,k}\, m_{j,k}$$

donde $j = |Q| - 1$, $k = |\Gamma| - 1$

Un vector *alphabet* de la forma:

$$a_0 a_1 a_2 \ldots a_{|\Sigma|-1} a_{|\Sigma|} a_{|\Sigma|+1} \ldots a_{|\Gamma|-2} a_{|\Gamma|-1}$$

donde los componentes $a_0 \ldots a_{|\Sigma|-1}$ corresponden a los elementos del conjunto Σ, los componentes $a_{|\Sigma|} \ldots a_{|\Gamma|-2}$ corresponden a $\Gamma - \Sigma - \{B\}$, y $a_{|\Gamma|-1}$ corresponde a B el blanco.

Un vector *tape* de tamaño Ω en el cual se encuentra codificada la cadena de entrada.

4.2.2 Diseño de los Operadores Genéticos.

Tomando en cuenta la considerable longitud del vector que representa a los elementos de la población, y con el fin de incrementar la eficiencia del algoritmo genético, se ha determinado descartar la utilización de un alfabeto binario en la codificación de la población. Por lo que, se sugiere que las componentes del vector sean codificadas a partir de un alfabeto de números enteros, cuyo valor máximo se encuentra determinado por la cardinalidad de los conjuntos Q y Γ.

Lo anterior, nos lleva a la consideración de que los operadores genéticos quedarán restringidos a operar sobre un vector cuyas componentes presentan una codificación de más alto nivel, que la basada en un alfabeto binario; con lo cual se descarta la generación de elementos de la población que no cumplen con el conjunto de restricciones impuestas por el problema. Bajo las consideraciones anteriores, la aplicación del operador de cruzamiento se lleva a cabo de la siguiente forma [Gold89]:

Antes del cruzamiento, se tienen dos vectores de la forma:

$$q^1_{0,0}\, a^1_{0,0}\, m^1_{0,0}\, q^1_{0,1}\, a^1_{0,1} m^1_{0,1} \ldots q^1_{1,0}\, a^1_{1,0} m^1_{1,0}\, q^1_{1,1}\, a^1_{1,1} m^1_{1,1} \ldots q^1_{j,k}\, a^1_{j,k} m^1_{j,k}$$

$$q^2_{0,0}\, a^2_{0,0}\, m^2_{0,0}\, q^2_{0,1}\, a^2_{0,1} m^2_{0,1} \ldots q^2_{1,0}\, a^2_{1,0} m^2_{1,0}\, q^2_{1,1}\, a^2_{1,1} m^2_{1,1} \ldots q^2_{j,k}\, a^2_{j,k} m^2_{j,k}$$

donde $j = |Q| - 1$, $k = |\Gamma| - 1$.

Se procede a la elección de una variable aleatoria discreta θ uniformemente distribuida en el intervalo $[0, j \times k]$, que representa el punto de cruzamiento. Antes de aplicar el operador es necesario llevar a cabo una transformación de los subíndices que etiquetan a las componentes de los vectores, de la siguiente manera:

$$q_0^1 a_1^1 m_2^1 q_3^1 a_4^1 m_5^1 \ldots q_{\theta-1}^1 a_\theta^1 m_{\theta+1}^1 \ldots q_{j\times k-2}^1 a_{j\times k-1}^1 m_{j\times k}^1$$

$$q_0^2 a_1^2 m_2^2 q_3^2 a_4^2 m_5^2 \ldots q_{\theta-1}^2 a_\theta^2 m_{\theta+1}^2 \ldots q_{j\times k-2}^2 a_{j\times k-1}^2 m_{j\times k}^2$$

Con lo cual procedemos a generar los dos descendientes:

$$q_0^1 a_1^1 m_2^1 q_3^1 a_4^1 m_5^1 \ldots q_{\theta-1}^1 a_\theta^1 m_{\theta+1}^2 \ldots q_{j\times k-2}^2 a_{j\times k-1}^2 m_{j\times k}^2$$

$$q_0^2 a_1^2 m_2^2 q_3^2 a_4^2 m_5^2 \ldots q_{\theta-1}^2 a_\theta^2 m_{\theta+1}^1 \ldots q_{j\times k-2}^1 a_{j\times k-1}^1 m_{j\times k}^1$$

El operador de mutación puede actuar sobre cualquiera de las componentes del vector, ya sea que se trate de un estado q, un símbolo a, o el movimiento de la cabeza m. La información del tipo de componente que ha sido elegida para mutación puede ser empleada para la generación de elementos de la población que satisfacen las restricciones del problema.

Por ejemplo, si elemento al que debe ser aplicado el operador de mutación es un estado, se procede a la elección de una variable aleatoria discreta uniformemente distribuida en el intervalo $[0, Q-1]$, con la cual se reemplaza la componente anterior. Cuando la componente corresponde a un símbolo se elige entonces una variable en el intervalo $[0, \Gamma - 1]$, para sustituirla. Por último, si la componente es un movimiento de la cabeza, se considera el intervalo $[0,1]$.

4.2.3 Determinación de la Función Objetivo.

Con el fin de mapear el problema al dominio de un algoritmo genético, se ha desarrollado una función objetivo. En el caso en que se considera a la máquina de Turing como aceptador de lenguajes se sugiere definir una función

proporcional al número de secuencias aceptadas y/o a la longitud de la subsecuencia explorada por la máquina. En el cálculo de funciones se propone utilizar la distancia de Hamming existente entre el resultado producido por la máquina y el resultado deseado.

Más concretamente, en la consideración del enfoque de aceptación de lenguajes, la entrada que se presenta al sistema es un conjunto de secuencias que se supone pertenecen al lenguaje definido por la máquina, y un conjunto de secuencias contra ejemplo. Cada vez que la máquina de Turing proporciona una respuesta correcta en relación a la aceptación de la secuencia, se propone que el valor de la función objetivo sea incrementado, de la siguiente manera.

Si la máquina de Turing aceptó la secuencia, entonces
    Si la secuencia debe pertenecer al lenguaje, entonces
        incrementa el valor de la función objetivo.

Si la máquina de Turing rechazó la secuencia, entonces
    Si la secuencia es un contra ejemplo, entonces
        incrementa el valor de la función objetivo.

4.2.4 Selección de Probabilidades para la Aplicación de Operadores Genéticos

En la mayoría de los casos prácticos las probabilidades de recombinación p_c y de mutación p_m que se emplean para controlar la aplicación de los operadores genéticos son del orden de $p_c \approx .6$ mientras que $p_m \approx 0.01$ [Gold89]. Sin embargo, estos parámetros quedan susceptibles a modificaciones, con el fin de incrementar la eficiencia del algoritmo genético.

4.3 Implementación Computacional del Modelo de Generación Automática de Máquinas de Turing.

La implementación computacional del sistema de generación automática de máquinas de Turing se apoya en el algoritmo genético desarrollado en el presente trabajo (véase apéndice A), y se han llevado a cabo las adaptaciones necesarias para construir el sistema **turing**.

Se ha añadido al sistema, un analizador de léxico y de sintaxis que permite la especificación (en un archivo) de los parámetros de la simulación con las siguientes especificaciones:

stringnumber [*número de secuencias*]

popsize [*tamaño de la población*]

generations [*número de generaciones*]

pc [*probabilidad de cruzamiento*]

pm [*probabilidad de mutación*]

C [*constante de reescalamiento*]

statenumber [*cardinalidad del conjunto* Q]

symbolnumber [*cardinalidad del conjunto* Γ]

inputsymbolnumber [*cardinalidad del conjunto* Σ]

alphabet [*alfabeto* Σ]

maxmoves [*valor de* Π]

tapesize [*valor de* Ω]

[*nombre del archivo* 1][*tamaño del archivo* 1][0,1]

[*nombre del archivo* 2][*tamaño del archivo* 2][0,1]

.

[*nombre del archivo stringnumber*][*tamaño del archivo stringnumber*][0,1]

El valor de cero o uno que sigue al tamaño del archivo especifica el tipo de secuencia que contiene el archivo (ejemplo o contra ejemplo).

Por ejemplo:

```
stringnumber 7
popsize 512
generations 256
pc 0.6
pm 0.01
C 2.0
statenumber 128
symbolnumber 16
inputsimbolnumber 4
```

```
alphabet acgt
maxmoves 65536
tapesize 16384
hiv2ben 10359 1
hiv2rod 9671 1
hiv2isy 9636 1
hiv2gh1 9480 1
hiv2d194 9398 1
hiv2st 9672 1
hiv2nihz 9431 1
```

Para la implementación del cálculo de direcciones en el vector de la población se definen las macros **tapesymbol** y **transitionoffset.**

```
#define tapesymbol (tape[tapepointer])
#define transitionoffset ((state*symbolnumber+tapesymbol)*3)
```

Se define el conjunto de parámetros del modelo de generación automática de máquinas de Turing de la siguiente manera:

```
unsigned outputflag;
char *outputfile;
 unsigned char inputsymbolnumber,symbolnumber,statenumber,
stringnumber,*stringtype,*alphabet;
 unsigned char **stringname;
 unsigned maxmoves,tapesize,*stringsize;
unsigned char **string;
unsigned char *tape;
unsigned tapepointer;
unsigned char mask[256];
unsigned char code[256];
```

Donde cada una de las variable define:

outputflag	especifica el destino de la salida del programa.
outputfile	apuntador al archivo de salida.
inputsymbolnumber	tamaño del alfabeto de entrada.
symbolnumber	tamaño del alfabeto de cinta.
statenumber	número de estados de la máquina de Turing.
stringnumber	número de secuencias de entrenamiento.
stringtype	un apuntador al vector que define el tipo de secuencias (ejemplo o contra ejemplo).
alphabet	apuntador al vector que contiene al alfabeto de entrada.
stringname	apuntador al nombre de los archivos.
maxmoves	número máximo de transiciones.
tapesize	tamaño de la cinta de entrada.

stringsyze	apuntador al vector que contiene el tamaño de los archivos.
string	apuntador a los vectores que contienen a las secuencias de entrada.
tape	apuntador al vector que representa a la cinta.
tapepointer	posición actual en la cinta.
mask, code	arreglos que facilitan la codificación de la cinta.

La rutina **startevolution** lleva a cabo la asignación de memoria a las estructuras de datos definidas anteriormente, la lectura de los archivos que contienen a las secuencias de entrenamiento e inicializa aleatoriamente a los elementos de la población.

```
void startevolution(void)
        {
        unsigned i,j,k;
        FILE *input;
        gensize=(statenumber*symbolnumber)*3;
        population=(unsigned char **)calloc(popsize,sizeof(unsigned char *));
        string=(unsigned char **)calloc(stringnumber,sizeof(unsigned char *));
        tape=(unsigned char *)calloc(tapesize,sizeof(unsigned char));
        if(population==NULL || string==NULL )
                error(MEMORYERROR,0);
        for(i=0;i<popsize;++i)
                {
                population[i]=(unsigned char *)calloc(gensize,sizeof(unsigned char));
                if(population[i]==NULL)
                        error(MEMORYERROR,1);
                for(j=0,k=0;j<gensize;++j,k=(k+1)%3)
                        {
                        switch(k)
                                {
                                case 0:
                                        population[i][j]=(unsigned char)random(statenumber);
                                        break;
                                case 1:
                                        population[i][j]=(unsigned char)random(symbolnumber);
                                        break;
                                default:
                                        population[i][j]=(unsigned char)random(2);
                                }
                        }
                }
        for(i=0;i<256;++i)
                mask[i]=code[i]=0;
        for(i=0;i<inputsymbolnumber;++i)
                        {
                        mask[alphabet[i]]=1;
                        code[alphabet[i]]=i;
                        }
```

```
for(i=0;i<stringnumber;++i)
        {
        input=fopen((char *)stringname[i],"rb");
        if(input==NULL)
                error(FILEERROR,1);
        string[i]=(unsigned char *)calloc(stringsize[i],sizeof(unsigned char));
        if(string[i]==NULL)
                error(MEMORYERROR,3);
        fread(string[i],sizeof(unsigned char),stringsize[i],input);
        for(j=0,k=0;j<stringsize[i];++j)
                {
                if(!mask[string[i][j]] )continue;
                string[i][k]=code[string[i][j]];
                ++k;
                }
        stringsize[i]=k;
        fclose(input);
        }
}
```

La evaluación de los elementos de la población se realiza por medio de la rutina **eval**, que efectúa la simulación de la máquina de Turing para cada una de las secuencias de entrenamiento.

```
float eval(unsigned char *gene)
        {
        unsigned i,j,counter,offset,move;
        unsigned state;
        float score=0;
        for(i=0;i<stringnumber;++i)
                {
                for(j=0;j<stringsize[i];++j)
                        tape[j]=string[i][j];
                for(;j<tapesize;++j)
                        tape[j]=symbolnumber-1;
                state=0;
                tapepointer=0;
                counter=0;
                while(counter<maxmoves)
                        {
                        if(state==statenumber-1)
                                {
                                if(stringtype[i])
                                                score+=1;
                                break;
                                }
                        offset=transitionoffset;
                        state=gene[offset];
                        tape[tapepointer]=gene[offset+1];
                        move=gene[offset+2];
                        if(move==0 && tapepointer==0 )
                                {
                                if(!stringtype[i])score+=1;
                                break;
```

```
                        }
                tapepointer=move==1?tapepointer+1:tapepointer-1;
                if(tapepointer>tapesize)break;
                ++counter;
                }
        }
    return score;
    }
```

El operador de mutación, como se mencionó anteriormente se implementa de manera diferente, las modificaciones a la rutina **copy** se muestran a continuación:

```
unsigned copy(unsigned c,unsigned k)
    {
    unsigned newc;
    if(event(pm))
        {
        switch(k)
            {
            case 0:
                k=statenumber;
                break;
            case 1:
                k=symbolnumber;
                break;
            default:
                k=2;
            }
        while(1)
            {
            newc=random(k);
            if(newc!=c)return newc;
            }
        }
    else
        return c;
    }
```

La rutina **printturing** presenta los resultados de la simulación en un archivo que contiene los parámetros principales del modelo y los mejores elementos de la población.

```
void printturing(void)
    {
    FILE *output;
    unsigned i,j;
    float maxobjective=0;
    if(outputflag)
        output=fopen(outputfile,"wb");
    else
        {
```

```
            system("clear");
            output=stdout;
            }
    for(i=0;i<popsize;++i)
            if(maxobjective<objective[i])
                    maxobjective=objective[i];
    fwrite(&stringnumber,sizeof(unsigned char),1,output);
    for(i=0;i<stringnumber;++i)
            fwrite(stringname[i],sizeof(unsigned char),10,output);
    fwrite(&statenumber,sizeof(unsigned char),1,output);
    fwrite(&symbolnumber,sizeof(unsigned char),1,output);
    fwrite(&inputsymbolnumber,sizeof(unsigned char),1,output);
    fwrite(alphabet,sizeof(unsigned char),inputsymbolnumber,output);
    fwrite(&tapesize,sizeof(unsigned),1,output);
    fwrite(&maxmoves,sizeof(unsigned),1,output);
    fwrite(&gensize,sizeof(unsigned),1,output);
    for(i=0;i<popsize;++i)
            {
            if(objective[i]<maxobjective)continue;
            fwrite(population[i],sizeof(unsigned char),gensize,output);
            }
    if(outputflag)
            fclose(output);
    }
```

Ante la necesidad de liberar los recursos utilizados por el programa, se agregan las siguientes instrucciones a la rutina **endevolution:**

```
void endevolution(void)
    {
    free(tape);
    for(i=0;i<stringnumber;++i)
            {
            free(string[i]);
            }
    free(string);
    free(outputfile);
    }
```

Finalmente el programa principal recibe dos parámetros: uno es el nombre del archivo que contiene la configuración de la simulación y el otro es el nombre del archivo que debe contener los resultados del programa.

```
void main(int argc,char *argv[])
    {
    unsigned i;
    unsigned inputsize;
    FILE *input;
    if(argc==1)
            {
            puts("syntax:turing sourcefile outputfile");
            exit(0);
            }
```

```
if(argc==3)
        {
        outputfile=(char *)calloc(strlen(argv[2])+1,sizeof(char));
        strcpy(outputfile,argv[2]);
        outputflag=1;
        }
buffer=(unsigned char *)calloc(2048,sizeof(unsigned char));
if(buffer==NULL)
        error(MEMORYERROR,5);
input=fopen(argv[1],"rb");
inputsize=fread(buffer,sizeof(char),65536,input);
buffer[inputsize]=(unsigned char)EOF;
startturing();
free(buffer);
evolve();
}
```

Capítulo 5

Resultados Experimentales

5.1 Introducción.

El enfoque de máquina de Turing como aceptador de lenguajes es considerado con el fin de encontrar un procedimiento automático de decisión para la determinación de membresía a un lenguaje formal.

Con el fin de mapear el problema al dominio de los algoritmos genéticos, una función objetivo proporcional al número de secuencias aceptadas por la máquina de Turing fue desarrollada. Los resultados experimentales demuestran la eficiencia y rápida convergencia del algoritmo genético.

La entrada que se presenta al sistema consta de una colección de secuencias que se supone pertenecen al lenguaje definido por la máquina de Turing, y otras que se supone no pertenecen (contra ejemplos). Cada vez que una máquina de Turing proporcione una respuesta correcta en relación a la aceptación de cada una de las secuencias, el valor de la función objetivo es incrementado.

El conjunto de secuencias de entrenamiento esta conformado por biosecuencias (DNA) del virus de HIV principalmente en su variedad HIV-2 (sida africano), así como secuencias contra ejemplo generadas por medio de procedimientos automáticos desarrollados para el presente trabajo de investigación (véase apéndice B, programas **counterex, counterex.prefix, counterex.stats**).

Numerosas simulaciones del sistema **turing** con diferentes parámetros y conjuntos de secuencias de entrenamiento han sido ejecutadas hasta el momento, por lo que es conveniente separarlas en casos para su exposición.

5.2 Caso 1.

En el caso más sencillo se consideró un conjunto de entrenamiento que contiene siete secuencias del virus de HIV-2 (hiv2ben, hiv2rod, hiv2isy, hiv2st, hiv2d194, hiv2nihz, hiv2gh1), para llevar a cabo la búsqueda de una máquina de Turing que aceptara todas las secuencias anteriores. La configuración de la simulación se muestra a continuación:

```
stringnumber 7
pm 0.01
pc 0.6
C 2.0
popsize 512
generations 256
statenumber 128
symbolnumber 16
tapesize 16384
inputsymbolnumber 4
alphabet acgt
maxmoves 65536
hiv2ben 10359 1
hiv2rod 9671 1
hiv2st 9672 1
hiv2isy 9636 1
hiv2d194 9398 1
hiv2gh1 9480 1
hiv2nihz 9431 1
```

Los resultados de la simulación fueron:

Valor máximo de la función objetivo:	7.
Valor alcanzado con el algoritmo genético:	6.

Los resultados pueden visualizarse mediante la salida del programa **sim** (véase apéndice B), que presenta la descripción instantánea que produce la simulación de la máquina de Turing, operando sobre cada una de las cintas que contienen a las secuencias de entrenamiento. Para este caso se tiene:

```
hiv2ben
|- q0 t g c a a g g g a t
Tape not accepted
hiv2d194
|- q0 a g t c g c t c t g
|- 13 q60 g t c g c t c t g c
|- 13 9 q76 t c g c t c t g c g
```

|- 13 9 5 q74 c g c t c t g c g g
|- 13 9 5 15 q80 g c t c t g c g g a
|- 13 9 5 q109 15 13 c t c t g c g g
|- 13 9 5 15 q37 13 c t c t g c g g a
|- 13 9 5 15 14 q21 c t c t g c g g a g
|- 13 9 5 15 14 15 q123 t c t g c g g a g a
|- 13 9 5 15 14 q9 15 a c t g c g g a g
|- 13 9 5 15 q126 14 g a c t g c g g a
|- 13 9 5 15 9 q120 g a c t g c g g a g
|- 13 9 5 15 9 7 q49 a c t g c g g a g a
|- 13 9 5 15 9 q78 7 15 c t g c g g a g
|- 13 9 5 15 9 a q6 15 c t g c g g a g a
|- 13 9 5 15 9 a a q61 c t g c g g a g a g
|- 13 9 5 15 9 a q18 a 15 t g c g g a g a
|- 13 9 5 15 9 a 8 q44 15 t g c g g a g a g
|- 13 9 5 15 9 a 8 7 q39 t g c g g a g a g g
|- 13 9 5 15 9 a 8 q41 7 c g c g g a g a g
|- 13 9 5 15 9 a q22 8 9 c g c g g a g a
|- 13 9 5 15 9 q79 a 15 9 c g c g g a g
|- 13 9 5 15 9 9 q30 15 9 c g c g g a g a
|- 13 9 5 15 9 9 6 q29 9 c g c g g a g a g
|- 13 9 5 15 9 9 q110 6 14 c g c g g a g a
|- 13 9 5 15 9 9 9 q76 14 c g c g g a g a g
|- 13 9 5 15 9 9 9 5 q10 c g c g g a g a g g
|- 13 9 5 15 9 9 9 q127 5 t g c g g a g a g
Tape accepted
hiv2gh1
|- q0 c a g t c g c t c t
|- 7 q42 a g t c g c t c t g
|- 7 6 q85 g t c g c t c t g g
|- 7 q127 6 4 t c g c t c t g
Tape accepted
hiv2isy
|- q0 a g t c g c t c t g
|- 13 q60 g t c g c t c t g c
|- 13 9 q76 t c g c t c t g c g
|- 13 9 5 q74 c g c t c t g c g g
|- 13 9 5 15 q80 g c t c t g c g g a
|- 13 9 5 q109 15 13 c t c t g c g g
|- 13 9 5 15 q37 13 c t c t g c g g a
|- 13 9 5 15 14 q21 c t c t g c g g a g
|- 13 9 5 15 14 15 q123 t c t g c g g a g a
|- 13 9 5 15 14 q9 15 a c t g c g g a g
|- 13 9 5 15 q126 14 g a c t g c g g a
|- 13 9 5 15 9 q120 g a c t g c g g a g
|- 13 9 5 15 9 7 q49 a c t g c g g a g a
|- 13 9 5 15 9 q78 7 15 c t g c g g a g
|- 13 9 5 15 9 a q6 15 c t g c g g a g a
|- 13 9 5 15 9 a a q61 c t g c g g a g a g
|- 13 9 5 15 9 a q18 a 15 t g c g g a g a
|- 13 9 5 15 9 a 8 q44 15 t g c g g a g a g
|- 13 9 5 15 9 a 8 7 q39 t g c g g a g a g g
|- 13 9 5 15 9 a 8 q41 7 c g c g g a g a g
|- 13 9 5 15 9 a q22 8 9 c g c g g a g a
|- 13 9 5 15 9 q79 a 15 9 c g c g g a g
|- 13 9 5 15 9 9 q30 15 9 c g c g g a g a

|- 13 9 5 15 9 9 6 q29 9 c g c g g a g a g
|- 13 9 5 15 9 9 q110 6 14 c g c g g a g a
|- 13 9 5 15 9 9 9 q76 14 c g c g g a g a g
|- 13 9 5 15 9 9 9 5 q10 c g c g g a g a g g
|- 13 9 5 15 9 9 9 q127 5 t g c g g a g a g
Tape accepted
hiv2nihz
|- q0 a g t c g c t c t g
|- 13 q60 g t c g c t c t g g
|- 13 9 q76 t c g c t c t g g c
|- 13 9 5 q74 c g c t c t g g c a
|- 13 9 5 15 q80 g c t c t g g c a g
|- 13 9 5 q109 15 13 c t c t g g c a
|- 13 9 5 15 q37 13 c t c t g g c a g
|- 13 9 5 15 14 q21 c t c t g g c a g a
|- 13 9 5 15 14 15 q123 t c t g g c a g a g
|- 13 9 5 15 14 q9 15 a c t g g c a g a
|- 13 9 5 15 q126 14 g a c t g g c a g
|- 13 9 5 15 9 q120 g a c t g g c a g a
|- 13 9 5 15 9 7 q49 a c t g g c a g a g
|- 13 9 5 15 9 q78 7 15 c t g g c a g a
|- 13 9 5 15 9 a q6 15 c t g g c a g a g
|- 13 9 5 15 9 a a q61 c t g g c a g a g g
|- 13 9 5 15 9 a q18 a 15 t g g c a g a g
|- 13 9 5 15 9 a 8 q44 15 t g g c a g a g g
|- 13 9 5 15 9 a 8 7 q39 t g g c a g a g g c
|- 13 9 5 15 9 a 8 q41 7 c g g c a g a g g
|- 13 9 5 15 9 a q22 8 9 c g g c a g a g
|- 13 9 5 15 9 q79 a 15 9 c g g c a g a
|- 13 9 5 15 9 9 q30 15 9 c g g c a g a g
|- 13 9 5 15 9 9 6 q29 9 c g g c a g a g g
|- 13 9 5 15 9 9 q110 6 14 c g g c a g a g
|- 13 9 5 15 9 9 9 q76 14 c g g c a g a g g
|- 13 9 5 15 9 9 9 5 q10 c g g c a g a g g c
|- 13 9 5 15 9 9 9 q127 5 t g g c a g a g g
Tape accepted
hiv2rod
|- q0 g g t c g c t c t g
|- g q109 g t c g c t c t g c
|- g t q79 t c g c t c t g c g
|- g q45 t 9 c g c t c t g c
|- g 5 q2 9 c g c t c t g c g
|- g q16 5 7 c g c t c t g c
|- g 10 q76 7 c g c t c t g c g
|- g 10 t q100 c g c t c t g c g g
|- g 10 t 15 q72 g c t c t g c g g a
|- g 10 t 15 15 q14 c t c t g c g g a g
|- g 10 t 15 15 13 q96 t c t g c g g a g a
|- g 10 t 15 15 q69 13 t c t g c g g a g
|- g 10 t 15 q127 15 13 t c t g c g g a
Tape accepted
hiv2st
|- q0 a g t c g c t c t g
|- 13 q60 g t c g c t c t g c
|- 13 9 q76 t c g c t c t g c g
|- 13 9 5 q74 c g c t c t g c g g

```
|- 13 9 5 15 q80 g c t c t g c g g a
|- 13 9 5 q109 15 13 c t c t g c g g
|- 13 9 5 15 q37 13 c t c t g c g g a
|- 13 9 5 15 14 q21 c t c t g c g g a g
|- 13 9 5 15 14 15 q123 t c t g c g g a g a
|- 13 9 5 15 14 q9 15 a c t g c g g a g
|- 13 9 5 15 q126 14 g a c t g c g g a
|- 13 9 5 15 9 q120 g a c t g c g g a g
|- 13 9 5 15 9 7 q49 a c t g c g g a g a
|- 13 9 5 15 9 q78 7 15 c t g c g g a g
|- 13 9 5 15 9 a q6 15 c t g c g g a g a
|- 13 9 5 15 9 a a q61 c t g c g g a g a g
|- 13 9 5 15 9 a q18 a 15 t g c g g a g a
|- 13 9 5 15 9 a 8 q44 15 t g c g g a g a g
|- 13 9 5 15 9 a 8 7 q39 t g c g g a g a g g
|- 13 9 5 15 9 a 8 q41 7 c g c g g a g a g
|- 13 9 5 15 9 a q22 8 9 c g c g g a g a
|- 13 9 5 15 9 q79 a 15 9 c g c g g a g
|- 13 9 5 15 9 9 q30 15 9 c g c g g a g a
|- 13 9 5 15 9 9 6 q29 9 c g c g g a g a g
|- 13 9 5 15 9 9 q110 6 14 c g c g g a g a
|- 13 9 5 15 9 9 9 q76 14 c g c g g a g a g
|- 13 9 5 15 9 9 9 5 q10 c g c g g a g a g g
|- 13 9 5 15 9 9 9 q127 5 t g c g g a g a g
Tape accepted
Strings accepted:
hiv2d194
hiv2gh1
hiv2isy
hiv2nihz
hiv2rod
hiv2st
Strings not accepted:
hiv2ben
```

5.3 Caso 2.

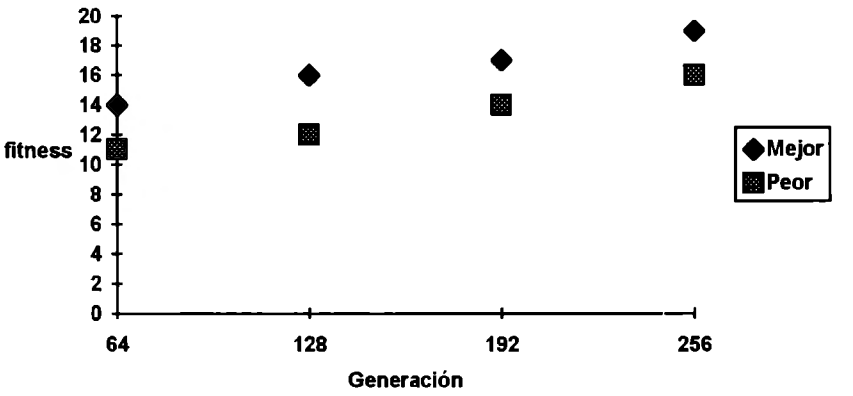
En el caso anterior únicamente ejemplos positivos fueron proporcionados al sistema, lo que significa que se proveen instancias del lenguaje definido por la máquina de Turing, sin embargo, no se proporciona información para prevenir la sobre generalización. La sobre generalización puede ser evitada mediante la consideración de ejemplos negativos (contra ejemplos) para inducir el lenguaje que describe a todas las secuencias ejemplo y ninguna secuencia contra ejemplo.

El conjunto de secuencias de entrenamiento para la simulación fue conformado por secuencias del virus de HIV-2 (hiv2ben, hiv2rod, hiv2st, hiv2isy, hiv2d194, hiv2gh1 y hiv2nihz), así como por dos contra ejemplo de cada una de las

anteriores. El propósito de la simulación es encontrar una máquina de Turing que acepta a todas las secuencias del virus de HIV-2 y rechaza a todos los contra ejemplos. Las secuencias contra ejemplo fueron generadas por el programa **counterex.** La configuración de la simulación para este caso fue:

```
stringnumber 21
pm 0.01
pc 0.6
C 2.0
popsize 512
generations 256
statenumber 128
symbolnumber 16
tapesize 16384
inputsymbolnumber 4
alphabet acgt
maxmoves 65536
hiv2ben 10359 1
hiv2benx 10359 0
hiv2beny 10359 0
hiv2rod 9671 1
hiv2rodx 9671 0
hiv2rody 9671 0
hiv2st 9672 1
hiv2stx 9672 0
hiv2sty 9672 0
hiv2isy 9636 1
hiv2isyx 9636 0
hiv2isyy 9636 0
hiv2d194 9398 1
hiv2d194x 9398 0
hiv2d194y 9398 0
hiv2gh1 9480 1
hiv2gh1x 9480 0
hiv2gh1y 9480 0
hiv2nihz 9431 1
hiv2nihzx 9431 0
hiv2nihzy 9431 0
```

Los resultados de la simulación se muestran en la siguiente gráfica:



Valor máximo de la función objetivo: 21.
Valor alcanzado con el algoritmo genético: 19.

Los resultados del programa **sim** se muestran a continuación:

```
hiv2ben
|- q0 t g c a a g g g a t
Tape not accepted
hiv2benx
|- q0 g a t c g g c a g g
Tape not accepted
hiv2beny
|- q0 g c a a a a c g c g
Tape not accepted
hiv2rod
|- q0 g g t c g c t c t g
Tape not accepted
hiv2rodx
|- q0 a g t c g t t g t t
|- a q52 g t c g t t g t t a
|- a g q108 t c g t t g t t a g
|- a g 13 q7 c g t t g t t a g a
|- a g q115 13 12 g t t g t t a g
|- a g 7 q98 12 g t t g t t a g a
|- a g 7 5 q77 g t t g t t a g a a
|- a g 7 5 6 q112 t t g t t a g a a g
|- a g 7 5 q43 6 9 t g t t a g a a
|- a g 7 q123 5 9 9 t g t t a g a
|- a g 7 5 q125 9 9 t g t t a g a a
|- a g 7 q110 5 8 9 t g t t a g a
|- a g q31 7 15 8 9 t g t t a g
|- a g t q120 15 8 9 t g t t a g a
|- a g q48 t 13 8 9 t g t t a g
|- a q73 g 9 13 8 9 t g t t a
|- q94 a g 9 13 8 9 t g t t
Tape not accepted
hiv2rody
|- q0 g c a g a g g a a a
```

Tape not accepted
hiv2st
|- q0 a g t c g c t c t g
|- a q52 g t c g c t c t g c
|- a g q108 t c g c t c t g c g
|- a g 13 q7 c g c t c t g c g g
|- a g q115 13 12 g c t c t g c g
|- a g 7 q98 12 g c t c t g c g g
|- a g 7 5 q77 g c t c t g c g g a
|- a g 7 5 6 q112 c t c t g c g g a g
|- a g 7 5 6 15 q14 t c t g c g g a g a
|- a g 7 5 6 q19 15 7 c t g c g g a g
|- a g 7 5 6 t q44 7 c t g c g g a g a
|- a g 7 5 6 t 12 q72 c t g c g g a g a g
|- a g 7 5 6 t 12 10 q48 t g c g g a g a g g
|- a g 7 5 6 t 12 q73 10 9 g c g g a g a g
|- a g 7 5 6 t q85 12 6 9 g c g g a g a
|- a g 7 5 6 q77 t 14 6 9 g c g g a g
|- a g 7 5 q54 6 8 14 6 9 g c g g a
|- a g 7 5 10 q89 8 14 6 9 g c g g a g
|- a g 7 5 q74 10 5 14 6 9 g c g g a
|- a g 7 5 6 q122 5 14 6 9 g c g g a g
|- a g 7 5 q37 6 13 14 6 9 g c g g a
|- a g 7 5 6 q88 13 14 6 9 g c g g a g
|- a g 7 5 6 14 q18 14 6 9 g c g g a g a
|- a g 7 5 6 q60 14 10 6 9 g c g g a g
|- a g 7 5 6 c q7 10 6 9 g c g g a g a
|- a g 7 5 6 q120 c 8 6 9 g c g g a g
|- a g 7 5 q109 6 12 8 6 9 g c g g a
|- a g 7 q40 5 11 12 8 6 9 g c g g
|- a g 7 g q12 11 12 8 6 9 g c g g a
|- a g 7 g 10 q56 12 8 6 9 g c g g a g
|- a g 7 g q99 10 6 8 6 9 g c g g a
|- a g 7 q108 g 7 6 8 6 9 g c g g
|- a g 7 12 q70 7 6 8 6 9 g c g g a
|- a g 7 q38 12 11 6 8 6 9 g c g g
|- a g q118 7 t 11 6 8 6 9 g c g
|- a q81 g 7 t 11 6 8 6 9 g c
|- a 14 q120 7 t 11 6 8 6 9 g c g
|- a 14 a q41 t 11 6 8 6 9 g c g g
|- a 14 a c q121 11 6 8 6 9 g c g g a
|- a 14 a q127 c 15 6 8 6 9 g c g g
Tape accepted
hiv2stx
|- q0 t c a t c a a t c c
Tape not accepted
hiv2sty
|- q0 a g c a a g a c a a
|- a q52 g c a a g a c a a t
|- a g q108 c a a g a c a a t a
|- a g 6 q125 a a g a c a a t a a
|- a g q18 6 7 a g a c a a t a
|- a q12 g 14 7 a g a c a a t
|- q115 a 11 14 7 a g a c a a
Tape not accepted
hiv2isy

|- q0 a g t c g c t c t g
|- a q52 g t c g c t c t g c
|- a g q108 t c g c t c t g c g
|- a g 13 q7 c g c t c t g c g g
|- a g q115 13 12 g c t c t g c g
|- a g 7 q98 12 g c t c t g c g g
|- a g 7 5 q77 g c t c t g c g g a
|- a g 7 5 6 q112 c t c t g c g g a g
|- a g 7 5 6 15 q14 t c t g c g g a g a
|- a g 7 5 6 q19 15 7 c t g c g g a g
|- a g 7 5 6 t q44 7 c t g c g g a g a
|- a g 7 5 6 t 12 q72 c t g c g g a g a g
|- a g 7 5 6 t 12 10 q48 t g c g g a g a g g
|- a g 7 5 6 t 12 q73 10 9 g c g g a g a g
|- a g 7 5 6 t q85 12 6 9 g c g g a g a
|- a g 7 5 6 q77 t 14 6 9 g c g g a g
|- a g 7 5 q54 6 8 14 6 9 g c g g a
|- a g 7 5 10 q89 8 14 6 9 g c g g a g
|- a g 7 5 q74 10 5 14 6 9 g c g g a
|- a g 7 5 6 q122 5 14 6 9 g c g g a g
|- a g 7 5 q37 6 13 14 6 9 g c g g a
|- a g 7 5 6 q88 13 14 6 9 g c g g a g
|- a g 7 5 6 14 q18 14 6 9 g c g g a g a
|- a g 7 5 6 q60 14 10 6 9 g c g g a g
|- a g 7 5 6 c q7 10 6 9 g c g g a g a
|- a g 7 5 6 q120 c 8 6 9 g c g g a g
|- a g 7 5 q109 6 12 8 6 9 g c g g a
|- a g 7 q40 5 11 12 8 6 9 g c g g
|- a g 7 g q12 11 12 8 6 9 g c g g a
|- a g 7 g 10 q56 12 8 6 9 g c g g a g
|- a g 7 g q99 10 6 8 6 9 g c g g a
|- a g 7 q108 g 7 6 8 6 9 g c g g
|- a g 7 12 q70 7 6 8 6 9 g c g g a
|- a g 7 q38 12 11 6 8 6 9 g c g g
|- a g q118 7 t 11 6 8 6 9 g c g
|- a q81 g 7 t 11 6 8 6 9 g c
|- a 14 q120 7 t 11 6 8 6 9 g c g
|- a 14 a q41 t 11 6 8 6 9 g c g g
|- a 14 a c q121 11 6 8 6 9 g c g g a
|- a 14 a q127 c 15 6 8 6 9 g c g g
Tape accepted
hiv2isyx
|- q0 t a t a t g g t g a
Tape not accepted
hiv2isyy
|- q0 t g t a a g g c a c
Tape not accepted
hiv2d194
|- q0 a g t c g c t c t g
|- a q52 g t c g c t c t g c
|- a g q108 t c g c t c t g c g
|- a g 13 q7 c g c t c t g c g g
|- a g q115 13 12 g c t c t g c g
|- a g 7 q98 12 g c t c t g c g g
|- a g 7 5 q77 g c t c t g c g g a
|- a g 7 5 6 q112 c t c t g c g g a g

|- a g 7 5 6 15 q14 t c t g c g g a g a
|- a g 7 5 6 q19 15 7 c t g c g g a g
|- a g 7 5 6 t q44 7 c t g c g g a g a
|- a g 7 5 6 t 12 q72 c t g c g g a g a g
|- a g 7 5 6 t 12 10 q48 t g c g g a g a g g
|- a g 7 5 6 t 12 q73 10 9 g c g g a g a g
|- a g 7 5 6 t q85 12 6 9 g c g g a g a
|- a g 7 5 6 q77 t 14 6 9 g c g g a g
|- a g 7 5 q54 6 8 14 6 9 g c g g a
|- a g 7 5 10 q89 8 14 6 9 g c g g a g
|- a g 7 5 q74 10 5 14 6 9 g c g g a
|- a g 7 5 6 q122 5 14 6 9 g c g g a g
|- a g 7 5 q37 6 13 14 6 9 g c g g a
|- a g 7 5 6 q88 13 14 6 9 g c g g a g
|- a g 7 5 6 14 q18 14 6 9 g c g g a g a
|- a g 7 5 6 q60 14 10 6 9 g c g g a g
|- a g 7 5 6 c q7 10 6 9 g c g g a g a
|- a g 7 5 6 q120 c 8 6 9 g c g g a g
|- a g 7 5 q109 6 12 8 6 9 g c g g a
|- a g 7 q40 5 11 12 8 6 9 g c g g
|- a g 7 g q12 11 12 8 6 9 g c g g a
|- a g 7 g 10 q56 12 8 6 9 g c g g a g
|- a g 7 g q99 10 6 8 6 9 g c g g a
|- a g 7 q108 g 7 6 8 6 9 g c g g
|- a g 7 12 q70 7 6 8 6 9 g c g g a
|- a g 7 q38 12 11 6 8 6 9 g c g g
|- a g q118 7 t 11 6 8 6 9 g c g
|- a q81 g 7 t 11 6 8 6 9 g c
|- a 14 q120 7 t 11 6 8 6 9 g c g
|- a 14 a q41 t 11 6 8 6 9 g c g g
|- a 14 a c q121 11 6 8 6 9 g c g g a
|- a 14 a q127 c 15 6 8 6 9 g c g g
Tape accepted
hiv2d194x
|- q0 c t a t c a g c c a
|- 13 q37 t a t c a g c c a c
|- q81 13 a a t c a g c c a
|- c q125 a a t c a g c c a c
|- q18 c 7 a t c a g c c a
|- 8 q82 7 a t c a g c c a c
|- 8 15 q23 a t c a g c c a c t
|- 8 q41 15 5 t c a g c c a c
|- 8 g q21 5 t c a g c c a c t
|- 8 q36 g 12 t c a g c c a c
|- 8 9 q21 12 t c a g c c a c t
|- 8 9 15 q113 t c a g c c a c t a
|- 8 9 q112 15 5 c a g c c a c t
|- 8 9 10 q42 5 c a g c c a c t a
|- 8 9 q92 10 4 c a g c c a c t
|- 8 q67 9 10 4 c a g c c a c
|- q70 8 11 10 4 c a g c c a
|- 4 q113 11 10 4 c a g c c a c
|- q116 4 7 10 4 c a g c c a
Tape not accepted
hiv2d194y
|- q0 t a g a g c t g a a

```
Tape not accepted
hiv2gh1
|- q0 c a g t c g c t c t
|- 13 q37 a g t c g c t c t g
|- 13 5 q68 g t c g c t c t g g
|- 13 q100 5 6 t c g c t c t g
|- 13 14 q8 6 t c g c t c t g g
|- 13 14 12 q127 t c g c t c t g g c
Tape accepted
hiv2gh1x
|- q0 t t t a a g g a g t
Tape not accepted
hiv2gh1y
|- q0 t a g a a a a a g g
Tape not accepted
hiv2nihz
|- q0 a g t c g c t c t g
|- a q52 g t c g c t c t g g
|- a g q108 t c g c t c t g g c
|- a g 13 q7 c g c t c t g g c a
|- a g q115 13 12 g c t c t g g c
|- a g 7 q98 12 g c t c t g g c a
|- a g 7 5 q77 g c t c t g g c a g
|- a g 7 5 6 q112 c t c t g g c a g a
|- a g 7 5 6 15 q14 t c t g g c a g a g
|- a g 7 5 6 q19 15 7 c t g g c a g a
|- a g 7 5 6 t q44 7 c t g g c a g a g
|- a g 7 5 6 t 12 q72 c t g g c a g a g g
|- a g 7 5 6 t 12 10 q48 t g g c a g a g g c
|- a g 7 5 6 t 12 q73 10 9 g g c a g a g g
|- a g 7 5 6 t q85 12 6 9 g g c a g a g
|- a g 7 5 6 q77 t 14 6 9 g g c a g a
|- a g 7 5 q54 6 8 14 6 9 g g c a g
|- a g 7 5 10 q89 8 14 6 9 g g c a g a
|- a g 7 5 q74 10 5 14 6 9 g g c a g
|- a g 7 5 6 q122 5 14 6 9 g g c a g a
|- a g 7 5 q37 6 13 14 6 9 g g c a g
|- a g 7 5 6 q88 13 14 6 9 g g c a g a
|- a g 7 5 6 14 q18 14 6 9 g g c a g a g
|- a g 7 5 6 q60 14 10 6 9 g g c a g a
|- a g 7 5 6 c q7 10 6 9 g g c a g a g
|- a g 7 5 6 q120 c 8 6 9 g g c a g a
|- a g 7 5 q109 6 12 8 6 9 g g c a g
|- a g 7 q40 5 11 12 8 6 9 g g c a
|- a g 7 g q12 11 12 8 6 9 g g c a g
|- a g 7 g 10 q56 12 8 6 9 g g c a g a
|- a g 7 g q99 10 6 8 6 9 g g c a g
|- a g 7 q108 g 7 6 8 6 9 g g c a
|- a g 7 12 q70 7 6 8 6 9 g g c a g
|- a g 7 q38 12 11 6 8 6 9 g g c a
|- a g q118 7 t 11 6 8 6 9 g g c
|- a q81 g 7 t 11 6 8 6 9 g g
|- a 14 q120 7 t 11 6 8 6 9 g g c
|- a 14 a q41 t 11 6 8 6 9 g g c a
|- a 14 a c q121 11 6 8 6 9 g g c a g
|- a 14 a q127 c 15 6 8 6 9 g g c a
```

```
Tape accepted
hiv2nihzx
|- q0 a g a g g a t t c g
|- a q52 g a g g a t t c g g
|- a g q108 a g g a t t c g g t
|- a q104 g 12 g g a t t c g g
|- q36 a 11 12 g g a t t c g
|- t q72 11 12 g g a t t c g g
|- t 6 q121 12 g g a t t c g g t
|- t q114 6 6 g g a t t c g g
|- q25 t t 6 g g a t t c g
|- 8 q82 t 6 g g a t t c g g
|- 8 a q45 6 g g a t t c g g t
|- 8 q98 a 13 g g a t t c g g
|- 8 t q1 13 g g a t t c g g t
|- 8 q26 t a g g a t t c g g
|- 8 10 q23 a g g a t t c g g t
|- 8 q41 10 5 g g a t t c g g
|- q103 8 13 5 g g a t t c g
Tape not accepted
hiv2nihzy
|- q0 a a a g g c a g g t
|- a q52 a a g g c a g g t a
|- a 4 q35 a g g c a g g t a t
|- a 4 9 q102 g g c a g g t a t c
|- a 4 q43 9 5 g c a g g t a t
|- a q95 4 4 5 g c a g g t a
|- q87 a t 4 5 g c a g g t
Tape not accepted
Strings accepted:
hiv2st
hiv2isy
hiv2d194
hiv2gh1
hiv2nihz
Strings not accepted:
hiv2ben
hiv2benx
hiv2beny
hiv2rod
hiv2rodx
hiv2rody
hiv2stx
hiv2sty
hiv2isyx
hiv2isyy
hiv2d194x
hiv2d194y
hiv2gh1x
hiv2gh1y
hiv2nihzx
hiv2nihzy
```

El valor de 19 fue alcanzado a partir de la generación 212, y no fue superado en ninguna de las generaciones subsecuentes. El máximo de la función objetivo no

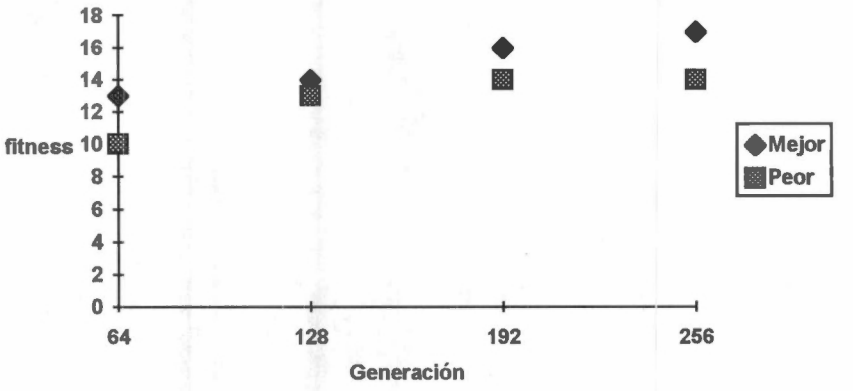
se consiguió debido a que las secuencias de HIV-2, hiv2ben y hiv2rod fueron rechazadas por las máquinas de Turing generadas mediante el algoritmo genético. Lo que permite hipotetizar acerca de ciertas diferencias de estas secuencias en relación a las cadenas del virus de HIV-2 que fueron aceptadas.

Cabe mencionar que en la simulación, la máquina de Turing sólo se emplea un prefijo muy pequeño de las secuencias de entrada. El prefijo mayor fue utilizado al llevar a cabo la simulación de la secuencia hiv2d194 y fue de 10 símbolos.

5.4 **Caso 3**.

En el presente caso, el entrenamiento del sistema **turing** consistió en presentar secuencias de HIV-2, así como secuencias contra ejemplo (generadas por el programa **counterex.prefix)** que contienen un prefijo de longitud N de la secuencias de HIV-2, con el fin de forzar a la máquina de Turing a emplear prefijos de longitud mayor durante su ejecución. Los parámetros referentes a las características de las máquinas de Turing y al algoritmo genético son idénticos al caso anterior. Los resultados se muestran a continuación.

Valor máximo de la función objetivo: 21.
Valor alcanzado con el algoritmo genético: 18.

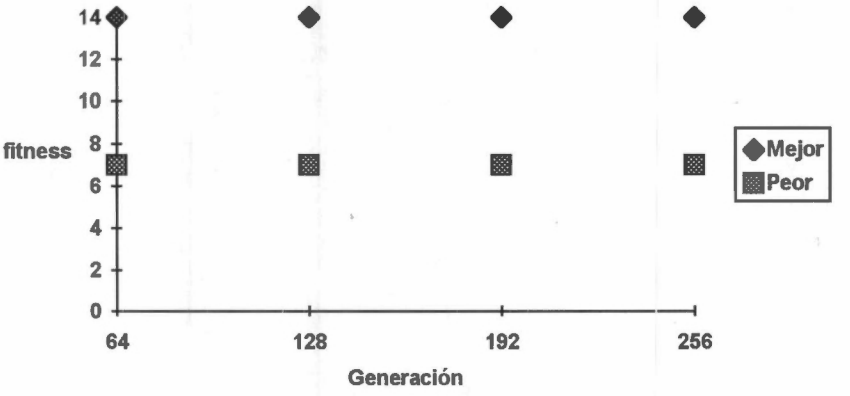


En la simulación el valor máximo de la función se alcanzó debido al rechazo de las cadenas hiv2ben, hiv2rod y a la aceptación de la cadena contraejemplo hiv2benx, sin embargo, la máquina de Turing utilizó un prefijo máximo del 4%

en la aceptación de la cadena hiv2st. Los resultados del sistema **sim** se omiten dada la longitud de los mismos.

Adicionalmente simulaciones con cadenas contra ejemplo conteniendo prefijos mayores (40,50 y 60 símbolos) de la cadena original se llevaron a cabo con los siguientes resultados:

Valor máximo de la función objetivo: 21.
Valor alcanzado con el algoritmo genético: 14.



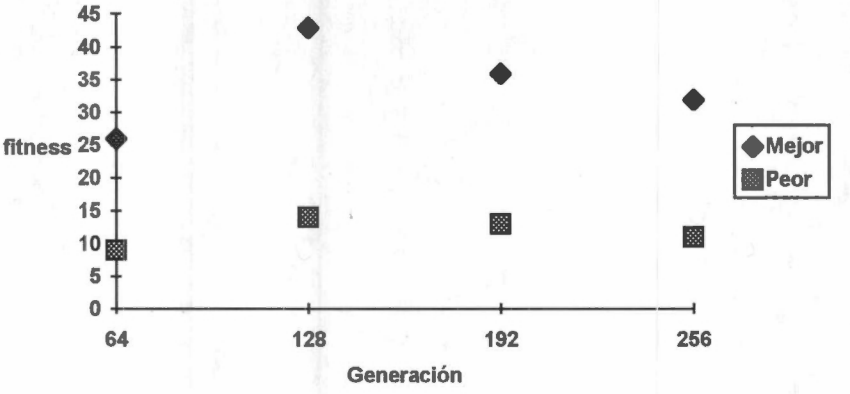
En esta simulación se alcanza el valor mayor de la función objetivo mediante el rechazo de las 14 secuencias contra ejemplo, el valor mínimo de la función se logra al aceptar las secuencias de HIV-2 y rechazar las secuencias contrajemplo. Lo que indica que en ningún caso fue posible aceptar secuencias de HIV-2 y fallar al aceptar sus correspondientes contra ejemplos.

5.5 Caso 4.

En la presente simulación cada secuencia de HIV-2 se presenta como entrada a la máquina de Turing en diferentes ocasiones y con diferentes posiciones de inicio aleatoriamente generadas, con el fin de explorar a las secuencias de una manera más detallada. Diferentes modificaciones al sistema **turing.c** se llevaron a cabo para hacer posible la simulación. Adicionalmente, se presentan secuencias contra ejemplo al sistema para la simulación. Los resultados son los siguientes:

Valor máximo de la función objetivo: 70.

Valor alcanzado con el algoritmo genético: 43.



Los resultados demuestran que debido a que las posiciones de inicio de la cadena son generadas estocásticamente y cambian en cada generación, la búsqueda de la solución se convierte en una caminata al azar, en la que los mejores elementos de una generación pueden tener un bajo desempeño en las subsecuentes.

5.6 Caso 5.

En la mayoría de los casos anteriores, se espera que la cabeza de lectura-escritura no tenga desplazamientos significativos hacia la derecha en la cinta, ésto se debe a que que la máquina de Turing inicia su ejecución con la cabeza de lectura-escritura posicionada en el extremo izquierdo de la cinta de entrada y que la probabilidad de generación del movimiento de la cabeza (izquierda o derecha) se lleva a cabo mediante la consideración de densidades equiprobables.

Con el fin de explorar las secuencias de HIV-2 de una manera más detallada, se propone llevar a cabo la simulación de la máquina de Turing colocando la cabeza de lectura-escritura en todas y cada una de las posiciones de las secuencias. En este caso el criterio de mérito es directamente proporcional a la longitud de las secuencias exploradas en la simulación de la máquina de Turing iniciando en cada posición de la secuencias de HIV-2, e inversamente proporcional a la longitud de las secuencias exploradas en los contra ejemplos.

Diferentes modificaciones fueron realizadas a los sistema **turing** y **sim** con el propósito de hacer posible la simulación del presente caso. Los resultados son representados por **sim** mediante la impresión de la secuencia explorada por la simulación de la máquina de Turing en cada posición. A continuación se presentan las secuencias exploradas al iniciar la máquina en las primeras 50 posiciones de cada una de las secuencias de HIV-2. La primera columna representa la posición de inicio de la máquina de Turing, la segunda representa la aceptación de la cinta (0,1), la tercera la longitud de la secuencia explorada, la cuarta y quinta la posición del extremo izquierdo y del extremo derecho de la secuencia respectivamente.

hiv2ben

0 0 1 0 0 t
1 1 6 1 6 gcaagg
2 0 3 0 2 tgc
3 0 4 0 3 tgca
4 1 3 2 4 caa
5 0 9 0 8 tgcaaggga
6 1 5 3 7 aaggg
7 0 12 0 11 tgcaagggatgt
8 0 9 0 8 tgcaaggga
9 0 10 0 9 tgcaagggat
10 0 13 0 12 tgcaagggatgtt
11 0 12 0 11 tgcaagggatgt
12 0 13 0 12 tgcaagggatgtt
13 0 14 0 13 tgcaagggatgttt
14 0 15 0 14 tgcaagggatgtttt
15 0 16 0 15 tgcaagggatgtttta
16 0 17 0 16 tgcaagggatgttttac
17 1 9 9 17 tgttttaca
18 1 4 16 19 cagt
19 0 20 0 19 tgcaagggatgttttacagt
20 1 19 2 20 caagggatgttttacagta
21 1 23 2 24 caagggatgttttacagtaggag
22 1 9 16 24 cagtaggag
23 0 24 0 23 tgcaagggatgttttacagtagga
24 0 28 0 27 tgcaagggatgttttacagtaggaggag
25 0 28 0 27 tgcaagggatgttttacagtaggaggag
26 0 27 0 26 tgcaagggatgttttacagtaggagga
27 0 32 0 31 tgcaagggatgttttacagtaggaggagacat
28 1 28 2 29 caagggatgttttacagtaggaggagac
29 0 30 0 29 tgcaagggatgttttacagtaggaggagac
30 1 10 21 30 ggaggagaca
31 0 32 0 31 tgcaagggatgttttacagtaggaggagacat
32 0 33 0 32 tgcaagggatgttttacagtaggaggagacata
33 0 36 0 35 tgcaagggatgttttacagtaggaggagacatagaa
34 1 5 30 34 ataga
35 0 36 0 35 tgcaagggatgttttacagtaggaggagacatagaa
36 1 7 30 36 atagaat
37 1 21 21 41 ggaggagacatagaatcctag

38 1 26 13 38 ttacagtaggaggagacatagaatcc
39 0 41 0 40 tgcaagggatgttttacagtaggaggagacatagaatccta
40 0 41 0 40 tgcaagggatgttttacagtaggaggagacatagaatccta
41 1 25 21 45 ggaggagacatagaatcctagacat
42 1 22 21 42 ggaggagacatagaatcctaga
43 1 23 21 43 ggaggagacatagaatcctagac
44 1 24 21 44 ggaggagacatagaatcctagaca
45 1 25 21 45 ggaggagacatagaatcctagacat
46 1 26 21 46 ggaggagacatagaatcctagacata
47 1 27 21 47 ggaggagacatagaatcctagacatat
48 1 28 21 48 ggaggagacatagaatcctagacatata
49 1 29 21 49 ggaggagacatagaatcctagacatatac

hiv2d194

0 0 1 0 0 a
1 0 3 0 2 agt
2 0 3 0 2 agt
3 0 9 0 8 agtcgctct
4 0 12 0 11 agtcgctctgcg
5 0 6 0 5 agtcgc
6 0 7 0 6 agtcgct
7 0 17 0 16 agtcgctctgcggagag
8 0 9 0 8 agtcgctct
9 1 10 3 12 cgctctgcgg
10 0 12 0 11 agtcgctctgcg
11 1 10 3 12 cgctctgcgg
12 1 8 10 17 cggagagg
13 0 14 0 13 agtcgctctgcgga
14 0 18 0 17 agtcgctctgcggagagg
15 1 8 9 16 gcggagag
16 1 13 6 18 tctgcggagaggc
17 1 18 3 20 cgctctgcggagaggctg
18 1 4 15 18 aggc
19 1 7 13 19 agaggct
20 0 22 0 21 agtcgctctgcggagaggctgg
21 0 31 0 30 agtcgctctgcggagaggctggcagattgag
22 0 23 0 22 agtcgctctgcggagaggctggc
23 0 24 0 23 agtcgctctgcggagaggctggca
24 1 10 20 29 ggcagattga
25 0 26 0 25 agtcgctctgcggagaggctggcaga
26 1 7 20 26 ggcagat
27 0 28 0 27 agtcgctctgcggagaggctggcagatt
28 1 4 27 30 tgag
29 1 9 22 30 cagattgag
30 1 20 13 32 agaggctggcagattgagcc
31 1 13 19 31 tggcagattgagc
32 1 6 27 32 tgagcc
33 1 5 29 33 agccc
34 1 17 19 35 tggcagattgagccctg
35 1 10 27 36 tgagccctgg
36 1 18 20 37 ggcagattgagccctggg
37 1 20 20 39 ggcagattgagccctgggag
38 1 20 19 38 tggcagattgagccctggga
39 1 25 19 43 tggcagattgagccctgggaggttc

40 1 9 33 41 ctgggaggt
41 1 23 19 41 tggcagattgagccctgggaggt
42 1 14 29 42 agccctgggaggtt
43 1 7 42 48 tctctcc
44 1 18 27 44 tgagccctgggaggttct
45 1 7 44 50 tctccag
46 1 13 34 46 tgggaggttctct
47 1 28 27 54 tgagccctgggaggttctctccagcact
48 1 30 19 48 tggcagattgagccctgggaggttctctcc
49 1 23 27 49 tgagccctgggaggttctctcca

hiv2gh1

0 0 1 0 0 c
1 0 2 0 1 ca
2 1 4 0 3 cagt
3 0 4 0 3 cagt
4 1 10 0 9 cagtcgctct
5 1 7 5 11 gctctgg
6 0 7 0 6 cagtcgc
7 1 8 0 7 cagtcgct
8 0 12 0 11 cagtcgctctgg
9 0 10 0 9 cagtcgctct
10 1 12 0 11 cagtcgctctgg
11 0 17 0 16 cagtcgctctggcagag
12 0 13 0 12 cagtcgctctggc
13 0 14 0 13 cagtcgctctggca
14 0 17 0 16 cagtcgctctggcagag
15 0 16 0 15 cagtcgctctggcaga
16 0 19 0 18 cagtcgctctggcagaggc
17 1 9 12 20 cagaggctg
18 1 4 15 18 aggc
19 1 7 13 19 agaggct
20 1 13 9 21 tggcagaggctgg
21 0 31 0 30 cagtcgctctggcagaggctggcagattgag
22 0 23 0 22 cagtcgctctggcagaggctggc
23 0 24 0 23 cagtcgctctggcagaggctggca
24 1 10 20 29 ggcagattga
25 1 16 10 25 ggcagaggctggcaga
26 1 7 20 26 ggcagat
27 1 18 10 27 ggcagaggctggcagatt
28 1 4 27 30 tgag
29 1 9 22 30 cagattgag
30 1 20 13 32 agaggctggcagattgagcc
31 1 13 19 31 tggcagattgagc
32 1 6 27 32 tgagcc
33 1 5 29 33 agccc
34 1 17 19 35 tggcagattgagccctg
35 1 10 27 36 tgagccctgg
36 1 18 20 37 ggcagattgagccctggg
37 1 20 20 39 ggcagattgagccctgggag
38 1 20 19 38 tggcagattgagccctggga
39 1 25 19 43 tggcagattgagccctgggaggttc
40 1 9 33 41 ctgggaggt
41 1 23 19 41 tggcagattgagccctgggaggt

42 1 14 29 42 agccctgggaggtt
43 1 7 42 48 tctctcc
44 1 18 27 44 tgagccctgggaggttct
45 1 7 44 50 tctccag
46 1 13 34 46 tgggaggttctct
47 1 28 27 54 tgagccctgggaggttctctccagcact
48 1 30 19 48 tggcagattgagccctgggaggttctctcc
49 1 23 27 49 tgagccctgggaggttctctcca

hiv2isy

0 0 1 0 0 a
1 0 3 0 2 agt
2 0 3 0 2 agt
3 0 9 0 8 agtcgctct
4 0 12 0 11 agtcgctctgcg
5 0 6 0 5 agtcgc
6 0 7 0 6 agtcgct
7 0 17 0 16 agtcgctctgcggagag
8 0 9 0 8 agtcgctct
9 1 10 3 12 cgctctgcgg
10 0 12 0 11 agtcgctctgcg
11 1 10 3 12 cgctctgcgg
12 1 8 10 17 cggagagg
13 0 14 0 13 agtcgctctgcgga
14 0 18 0 17 agtcgctctgcggagagg
15 1 8 9 16 gcggagag
16 1 13 6 18 tctgcggagaggc
17 1 18 3 20 cgctctgcggagaggctg
18 1 4 15 18 aggc
19 1 7 13 19 agaggct
20 0 22 0 21 agtcgctctgcggagaggctgg
21 0 31 0 30 agtcgctctgcggagaggctggcagattgag
22 0 23 0 22 agtcgctctgcggagaggctggc
23 0 24 0 23 agtcgctctgcggagaggctggca
24 1 10 20 29 ggcagattga
25 0 26 0 25 agtcgctctgcggagaggctggcaga
26 1 7 20 26 ggcagat
27 0 28 0 27 agtcgctctgcggagaggctggcagatt
28 1 4 27 30 tgag
29 1 9 22 30 cagattgag
30 1 20 13 32 agaggctggcagattgagcc
31 1 13 19 31 tggcagattgagc
32 1 6 27 32 tgagcc
33 1 5 29 33 agccc
34 1 17 19 35 tggcagattgagccctg
35 1 10 27 36 tgagccctgg
36 1 18 20 37 ggcagattgagccctggg
37 1 20 20 39 ggcagattgagccctgggag
38 1 20 19 38 tggcagattgagccctggga
39 1 25 19 43 tggcagattgagccctgggaggttc
40 1 9 33 41 ctgggaggt
41 1 23 19 41 tggcagattgagccctgggaggt
42 1 14 29 42 agccctgggaggtt
43 1 7 42 48 tctctcc

44 1 18 27 44 tgagccctgggaggttct
45 1 7 44 50 tctccag
46 1 13 34 46 tgggaggttctct
47 1 28 27 54 tgagccctgggaggttctctccagcact
48 1 30 19 48 tggcagattgagccctgggaggttctctcc
49 1 23 27 49 tgagccctgggaggttctctcca

hiv2nihz

0 0 1 0 0 a
1 0 3 0 2 agt
2 0 3 0 2 agt
3 0 9 0 8 agtcgctct
4 1 7 4 10 gctctgg
5 0 6 0 5 agtcgc
6 0 7 0 6 agtcgct
7 0 11 0 10 agtcgctctgg
8 0 9 0 8 agtcgctct
9 0 11 0 10 agtcgctctgg
10 0 16 0 15 agtcgctctggcagag
11 0 12 0 11 agtcgctctggc
12 0 13 0 12 agtcgctctggca
13 0 16 0 15 agtcgctctggcagag
14 0 15 0 14 agtcgctctggcaga
15 0 18 0 17 agtcgctctggcagaggc
16 1 9 11 19 cagaggctg
17 1 4 14 17 aggc
18 1 7 12 18 agaggct
19 1 13 8 20 tggcagaggctgg
20 0 30 0 29 agtcgctctggcagaggctggcagattgag
21 0 22 0 21 agtcgctctggcagaggctggc
22 0 23 0 22 agtcgctctggcagaggctggca
23 1 10 19 28 ggcagattga
24 1 16 9 24 ggcagaggctggcaga
25 1 7 19 25 ggcagat
26 1 18 9 26 ggcagaggctggcagatt
27 1 4 26 29 tgag
28 1 9 21 29 cagattgag
29 1 20 12 31 agaggctggcagattgagcc
30 1 13 18 30 tggcagattgagc
31 1 6 26 31 tgagcc
32 1 5 28 32 agccc
33 1 17 18 34 tggcagattgagccctg
34 1 10 26 35 tgagccctgg
35 1 18 19 36 ggcagattgagccctggg
36 1 20 19 38 ggcagattgagccctgggag
37 1 20 18 37 tggcagattgagccctggga
38 1 25 18 42 tggcagattgagccctgggaggttc
39 1 9 32 40 ctgggaggt
40 1 23 18 40 tggcagattgagccctgggaggt
41 1 14 28 41 agccctgggaggtt
42 1 7 41 47 tctctcc
43 1 18 26 43 tgagccctgggaggttct
44 1 7 43 49 tctccag
45 1 13 33 45 tgggaggttctct

46 1 28 26 53 tgagccctgggaggttctctccagcact
47 1 30 18 47 tggcagattgagccctgggaggttctctcc
48 1 23 26 48 tgagccctgggaggttctctcca
49 1 29 26 54 tgagccctgggaggttctctccagcacta

hiv2rod

0 0 2 0 1 gg
1 0 3 0 2 ggt
2 0 3 0 2 ggt
3 0 9 0 8 ggtcgctct
4 0 12 0 11 ggtcgctctgcg
5 0 6 0 5 ggtcgc
6 0 7 0 6 ggtcgct
7 0 17 0 16 ggtcgctctgcggagag
8 0 9 0 8 ggtcgctct
9 1 10 3 12 cgctctgcgg
10 0 12 0 11 ggtcgctctgcg
11 1 10 3 12 cgctctgcgg
12 1 8 10 17 cggagagg
13 0 14 0 13 ggtcgctctgcgga
14 1 18 0 17 ggtcgctctgcggagagg
15 1 8 9 16 gcggagag
16 1 13 6 18 tctgcggagaggc
17 1 18 3 20 cgctctgcggagaggctg
18 1 4 15 18 aggc
19 1 7 13 19 agaggct
20 0 22 0 21 ggtcgctctgcggagaggctgg
21 0 31 0 30 ggtcgctctgcggagaggctggcagattgag
22 0 23 0 22 ggtcgctctgcggagaggctggc
23 0 24 0 23 ggtcgctctgcggagaggctggca
24 1 10 20 29 ggcagattga
25 0 26 0 25 ggtcgctctgcggagaggctggcaga
26 1 7 20 26 ggcagat
27 0 28 0 27 ggtcgctctgcggagaggctggcagatt
28 1 4 27 30 tgag
29 1 9 22 30 cagattgag
30 1 20 13 32 agaggctggcagattgagcc
31 1 13 19 31 tggcagattgagc
32 1 6 27 32 tgagcc
33 1 5 29 33 agccc
34 1 17 19 35 tggcagattgagccctg
35 1 10 27 36 tgagccctgg
36 1 18 20 37 ggcagattgagccctggg
37 1 20 20 39 ggcagattgagccctgggag
38 1 20 19 38 tggcagattgagccctggga
39 1 25 19 43 tggcagattgagccctgggaggttc
40 1 9 33 41 ctgggaggt
41 1 23 19 41 tggcagattgagccctgggaggt
42 1 14 29 42 agccctgggaggtt
43 1 7 42 48 tctctcc
44 1 18 27 44 tgagccctgggaggttct
45 1 7 44 50 tctccag
46 1 13 34 46 tgggaggttctct
47 1 28 27 54 tgagccctgggaggttctctccagcact

48 1 30 19 48 tggcagattgagccctgggaggttctctcc
49 1 23 27 49 tgagccctgggaggttctctcca

hiv2st

0 0 1 0 0 a
1 0 3 0 2 agt
2 0 3 0 2 agt
3 0 9 0 8 agtcgctct
4 0 12 0 11 agtcgctctgcg
5 0 6 0 5 agtcgc
6 0 7 0 6 agtcgct
7 0 17 0 16 agtcgctctgcggagag
8 0 9 0 8 agtcgctct
9 1 10 3 12 cgctctgcgg
10 0 12 0 11 agtcgctctgcg
11 1 10 3 12 cgctctgcgg
12 1 8 10 17 cggagagg
13 0 14 0 13 agtcgctctgcgga
14 0 18 0 17 agtcgctctgcggagagg
15 1 8 9 16 gcggagag
16 1 13 6 18 tctgcggagaggc
17 1 18 3 20 cgctctgcggagaggctg
18 1 4 15 18 aggc
19 1 7 13 19 agaggct
20 0 22 0 21 agtcgctctgcggagaggctgg
21 0 31 0 30 agtcgctctgcggagaggctggcagattgag
22 0 23 0 22 agtcgctctgcggagaggctggc
23 0 24 0 23 agtcgctctgcggagaggctggca
24 1 10 20 29 ggcagattga
25 0 26 0 25 agtcgctctgcggagaggctggcaga
26 1 7 20 26 ggcagat
27 0 28 0 27 agtcgctctgcggagaggctggcagatt
28 1 4 27 30 tgag
29 1 9 22 30 cagattgag
30 1 20 13 32 agaggctggcagattgagcc
31 1 13 19 31 tggcagattgagc
32 1 6 27 32 tgagcc
33 1 5 29 33 agccc
34 1 17 19 35 tggcagattgagccctg
35 1 10 27 36 tgagccctgg
36 1 18 20 37 ggcagattgagccctggg
37 1 20 20 39 ggcagattgagccctgggag
38 1 20 19 38 tggcagattgagccctggga
39 1 25 19 43 tggcagattgagccctgggaggttc
40 1 9 33 41 ctgggaggt
41 1 23 19 41 tggcagattgagccctgggaggt
42 1 14 29 42 agccctgggaggtt
43 1 7 42 48 tctctcc
44 1 18 27 44 tgagccctgggaggttct
45 1 7 44 50 tctccag
46 1 13 34 46 tgggaggttctct
47 1 28 27 54 tgagccctgggaggttctctccagcact
48 1 30 19 48 tggcagattgagccctgggaggttctctcc
49 1 23 27 49 tgagccctgggaggttctctcca

Los resultados de la simulación demuestran que es posible llevar a cabo la alineación de secuencias exploradas mediante el sistema **turing**, similares a los realizados por O. Chavoya [Chav94], consiguendo de esta manera un consenso de las secuencias del virus de HIV-2, útil en el análisis de regularidades en las mismas.

Cabe hacer mención que el sistema **turing** ha descubierto una máquina de Turing de 32 estados y con alfabeto de cinta de ocho símbolos que divide las secuencias en segmentos de hasta tres mil símbolos aproximadamente. Los resultados de la simulación no se imprimen por razones obvias. La máquina en cuestión es la siguiente:

(q0, 0)= (q5, 0, R)
(q0, 1)= (q10, 3, L)
(q0, 2)= (q7, 5, L)
(q0, 3)= (q27, 5, L)
(q0, 4)= (q3, 6, R)
(q0, 5)= (q5, 4, R)
(q0, 6)= (q5, 1, R)
(q0, 7)= (q5, 3, L)
(q1, 0)= (q23, 1, R)
(q1, 1)= (q22, 3, R)
(q1, 2)= (q24, 0, R)
(q1, 3)= (q15, 2, R)
(q1, 4)= (q11, 3, L)
(q1, 5)= (q2, 6, R)
(q1, 6)= (q12, 4, R)
(q1, 7)= (q9, 4, L)
(q2, 0)= (q28, 6, L)
(q2, 1)= (q8, 7, R)
(q2, 2)= (q26, 3, R)
(q2, 3)= (q10, 1, R)
(q2, 4)= (q2, 4, R)
(q2, 5)= (q0, 0, R)
(q2, 6)= (q9, 7, L)
(q2, 7)= (q24, 6, R)
(q3, 0)= (q9, 3, R)
(q3, 1)= (q16, 1, R)
(q3, 2)= (q25, 5, R)
(q3, 3)= (q0, 4, R)
(q3, 4)= (q10, 1, L)
(q3, 5)= (q1, 3, L)
(q3, 6)= (q25, 2, L)
(q3, 7)= (q15, 7, L)
(q4, 0)= (q4, 6, L)
(q4, 1)= (q10, 5, L)
(q4, 2)= (q25, 5, L)
(q4, 3)= (q29, 0, L)
(q4, 4)= (q26, 3, L)
(q4, 5)= (q19, 3, R)
(q4, 6)= (q5, 2, L)
(q4, 7)= (q27, 5, L)
(q5, 0)= (q19, 0, L)
(q5, 1)= (q3, 5, R)
(q5, 2)= (q26, 0, L)
(q5, 3)= (q14, 4, R)
(q5, 4)= (q23, 1, R)
(q5, 5)= (q6, 4, R)
(q5, 6)= (q14, 5, R)
(q5, 7)= (q3, 5, L)
(q6, 0)= (q25, 7, R)
(q6, 1)= (q29, 4, R)
(q6, 2)= (q20, 4, R)
(q6, 3)= (q3, 6, R)
(q6, 4)= (q21, 1, R)
(q6, 5)= (q20, 3, L)
(q6, 6)= (q11, 7, L)
(q6, 7)= (q12, 2, L)
(q7, 0)= (q30, 7, R)
(q7, 1)= (q12, 1, L)
(q7, 2)= (q5, 0, R)
(q7, 3)= (q16, 7, R)
(q7, 4)= (q13, 0, R)
(q7, 5)= (q18, 7, R)
(q7, 6)= (q17, 0, L)
(q7, 7)= (q21, 3, R)
(q8, 0)= (q20, 6, R)
(q8, 1)= (q9, 0, R)
(q8, 2)= (q18, 7, L)
(q8, 3)= (q10, 5, R)
(q8, 4)= (q13, 2, L)
(q8, 5)= (q22, 5, R)
(q8, 6)= (q22, 4, L)
(q8, 7)= (q8, 3, R)
(q9, 0)= (q19, 5, L)
(q9, 1)= (q12, 0, R)
(q9, 2)= (q17, 5, L)
(q9, 3)= (q20, 3, L)
(q9, 4)= (q28, 6, R)
(q9, 5)= (q13, 3, L)
(q9, 6)= (q23, 5, R)
(q9, 7)= (q15, 3, R)
(q10, 0)= (q9, 0, L)
(q10, 1)= (q19, 7, R)
(q10, 2)= (q18, 7, R)
(q10, 3)= (q18, 1, L)
(q10, 4)= (q5, 3, R)
(q10, 5)= (q20, 2, R)
(q10, 6)= (q8, 3, L)
(q10, 7)= (q20, 3, L)
(q11, 0)= (q4, 0, L)
(q11, 1)= (q12, 5, R)
(q11, 2)= (q16, 1, L)
(q11, 3)= (q23, 0, R)
(q11, 4)= (q12, 4, R)
(q11, 5)= (q12, 0, L)
(q11, 6)= (q18, 7, R)
(q11, 7)= (q0, 6, L)
(q12, 0)= (q8, 2, R)
(q12, 1)= (q1, 7, R)
(q12, 2)= (q5, 0, L)
(q12, 3)= (q27, 2, R)
(q12, 4)= (q11, 6, R)
(q12, 5)= (q29, 6, L)
(q12, 6)= (q30, 2, L)
(q12, 7)= (q21, 4, R)
(q13, 0)= (q5, 2, R)
(q13, 1)= (q30, 6, L)
(q13, 2)= (q24, 3, L)
(q13, 3)= (q16, 1, R)
(q13, 4)= (q26, 5, L)
(q13, 5)= (q27, 4, L)
(q13, 6)= (q31, 6, R)

(q13, 7)= (q20, 7, R)
(q14, 0)= (q23, 2, R)
(q14, 1)= (q21, 2, R)
(q14, 2)= (q29, 4, L)
(q14, 3)= (q5, 6, L)
(q14, 4)= (q26, 2, R)
(q14, 5)= (q17, 7, R)
(q14, 6)= (q6, 1, L)
(q14, 7)= (q24, 0, L)
(q15, 0)= (q13, 1, L)
(q15, 1)= (q22, 4, R)
(q15, 2)= (q24, 1, R)
(q15, 3)= (q16, 5, R)
(q15, 4)= (q3, 0, L)
(q15, 5)= (q13, 5, R)
(q15, 6)= (q16, 7, R)
(q15, 7)= (q24, 5, L)
(q16, 0)= (q6, 4, L)
(q16, 1)= (q28, 2, L)
(q16, 2)= (q5, 2, R)
(q16, 3)= (q17, 1, L)
(q16, 4)= (q17, 6, L)
(q16, 5)= (q7, 3, R)
(q16, 6)= (q7, 3, L)
(q16, 7)= (q9, 1, R)
(q17, 0)= (q24, 3, L)
(q17, 1)= (q19, 1, R)
(q17, 2)= (q0, 3, L)
(q17, 3)= (q16, 1, R)
(q17, 4)= (q28, 1, R)
(q17, 5)= (q20, 6, L)
(q17, 6)= (q3, 7, R)
(q17, 7)= (q26, 7, R)
(q18, 0)= (q8, 6, R)
(q18, 1)= (q16, 3, R)
(q18, 2)= (q19, 2, L)
(q18, 3)= (q28, 0, R)
(q18, 4)= (q28, 7, L)
(q18, 5)= (q1, 7, R)
(q18, 6)= (q12, 3, R)
(q18, 7)= (q23, 7, R)
(q19, 0)= (q13, 2, R)
(q19, 1)= (q8, 5, L)
(q19, 2)= (q22, 6, R)
(q19, 3)= (q16, 2, R)
(q19, 4)= (q14, 5, L)
(q19, 5)= (q30, 7, L)
(q19, 6)= (q3, 3, R)
(q19, 7)= (q17, 3, R)
(q20, 0)= (q2, 0, L)
(q20, 1)= (q0, 4, L)
(q20, 2)= (q17, 0, R)
(q20, 3)= (q2, 1, L)
(q20, 4)= (q29, 0, L)
(q20, 5)= (q7, 5, L)
(q20, 6)= (q22, 2, L)
(q20, 7)= (q7, 2, R)
(q21, 0)= (q22, 7, R)
(q21, 1)= (q0, 1, L)
(q21, 2)= (q21, 7, R)
(q21, 3)= (q3, 5, R)
(q21, 4)= (q10, 7, L)
(q21, 5)= (q3, 6, L)
(q21, 6)= (q23, 0, L)
(q21, 7)= (q15, 5, R)
(q22, 0)= (q5, 1, R)
(q22, 1)= (q10, 4, R)
(q22, 2)= (q24, 0, L)
(q22, 3)= (q24, 4, R)
(q22, 4)= (q30, 2, R)
(q22, 5)= (q25, 3, L)
(q22, 6)= (q23, 1, L)
(q22, 7)= (q25, 5, L)
(q23, 0)= (q3, 4, R)
(q23, 1)= (q30, 7, L)
(q23, 2)= (q17, 0, R)
(q23, 3)= (q6, 5, R)
(q23, 4)= (q13, 7, R)
(q23, 5)= (q13, 5, L)
(q23, 6)= (q19, 3, R)
(q23, 7)= (q18, 2, L)
(q24, 0)= (q3, 1, R)
(q24, 1)= (q22, 6, R)
(q24, 2)= (q1, 1, R)
(q24, 3)= (q20, 3, R)
(q24, 4)= (q25, 4, L)
(q24, 5)= (q21, 2, L)
(q24, 6)= (q21, 1, L)
(q24, 7)= (q8, 7, R)
(q25, 0)= (q6, 0, R)
(q25, 1)= (q9, 4, L)
(q25, 2)= (q8, 5, R)
(q25, 3)= (q20, 1, R)
(q25, 4)= (q1, 1, R)
(q25, 5)= (q5, 7, L)
(q25, 6)= (q17, 2, L)
(q25, 7)= (q11, 3, L)
(q26, 0)= (q23, 0, R)
(q26, 1)= (q8, 0, R)
(q26, 2)= (q16, 4, L)
(q26, 3)= (q29, 6, R)
(q26, 4)= (q11, 5, R)
(q26, 5)= (q14, 2, R)
(q26, 6)= (q10, 2, R)
(q26, 7)= (q7, 1, R)
(q27, 0)= (q20, 7, L)
(q27, 1)= (q7, 2, R)
(q27, 2)= (q24, 5, R)
(q27, 3)= (q11, 4, R)
(q27, 4)= (q6, 2, R)
(q27, 5)= (q29, 2, R)
(q27, 6)= (q10, 6, R)
(q27, 7)= (q3, 1, R)
(q28, 0)= (q18, 7, L)
(q28, 1)= (q1, 6, R)
(q28, 2)= (q13, 6, R)
(q28, 3)= (q20, 7, L)
(q28, 4)= (q8, 1, R)
(q28, 5)= (q26, 4, R)
(q28, 6)= (q5, 5, R)
(q28, 7)= (q3, 5, L)
(q29, 0)= (q16, 0, R)
(q29, 1)= (q21, 1, R)
(q29, 2)= (q8, 7, L)
(q29, 3)= (q9, 5, L)
(q29, 4)= (q6, 7, L)
(q29, 5)= (q10, 5, R)
(q29, 6)= (q22, 1, R)
(q29, 7)= (q6, 6, L)
(q30, 0)= (q5, 4, R)
(q30, 1)= (q23, 4, R)
(q30, 2)= (q2, 4, L)
(q30, 3)= (q23, 5, R)
(q30, 4)= (q2, 4, R)
(q30, 5)= (q1, 4, L)
(q30, 6)= (q23, 7, L)
(q30, 7)= (q8, 4, L)
(q31, 0)= (q25, 1, R)
(q31, 1)= (q14, 7, L)
(q31, 2)= (q2, 7, L)
(q31, 3)= (q9, 0, R)
(q31, 4)= (q18, 1, L)
(q31, 5)= (q2, 0, L)
(q31, 6)= (q14, 6, L)
(q31, 7)= (q19, 7, R)

Capítulo 6

Conclusiones

Por medio del presente trabajo de investigación se propone un modelo evolutivo de máquinas de Turing como sistema de programación automática. Un modelo de algoritmos genéticos fue desarrollado, así mismo, una versión aumentada de la máquina de Turing fue desarrollada y utilizada para el reconocimiento de secuencias del virus HIV en su variedad HIV-2. Antes de sumarizar el modelo y su aplicabilidad a este dominio, se hace hincapié sobre la generalidad y poder del modelo propuesto.

La máquina de Turing constituye un poderoso dispositivo de la computación que nos permite modelar cualquier solución algorítmica a un problema. Es por lo tanto, un modelo viable para los sistemas de programación automática. El espacio de búsqueda que definen los posibles estados internos de una máquina de Turing puede ser fácilmente explorado mediante el uso de algoritmos genéticos.

El algoritmo genético desarrollado, implementa mecanismos que evitan la generación de elementos de la población que no cumplen con el conjunto de restricciones impuestas por el problema. Lo anterior nos lleva a un incremento de eficiencia y a la rápida convergencia del algoritmo.

El modelo se aplicó a la búsqueda de un lenguaje (definido por una máquina de Turing) que describiera a secuencias del virus HIV-2. En el caso 1 se encontró una máquina de Turing que aceptó a casi todas (sólo hiv2ben no fue aceptada) las secuencias del virus HIV-2, sin embargo, se trataba de una máquina trivial que aceptaba de igual manera a cadenas que no pertenecían al conjunto de entrenamiento.

En el caso 2 se llevó a cabo el entrenamiento del sistema adicionando un conjunto de secuencias contra ejemplo con el fin de evitar el problema de la sobre generalización (aceptar cualquier cadena). En este caso la máquina de Turing resultante una vez puesta en producción logró aceptar cadenas del virus HIV-1 (sida americano) y rechazar cadenas contra ejemplo que no se utilizaron durante el entrenamiento del sistema.

En el caso 3 el conjunto de secuencias contra ejemplo fueron forzadas a conservar en su estructura, prefijos de las secuencias del virus HIV-2, con el fin de que la máquina de Turing explorara mayor parte de la cinta durante su ejecución, lo cual se alcanzó mientras se utilizaron prefijos de longitud menor a 40 símbolos.

En el caso 5 se demostró que es posible encontrar una máquina de Turing que permite llevar a cabo la alineación de las secuencias del virus HIV, es decir, localizar segmentos que son comunes a todas las secuencias, con el uso mínimo de recursos de cómputo.

En todos los casos se obtuvieron resultados que permitieron desarrollar hipótesis acerca de regularidades e irregularidades en las secuencias, lo cual representa una importante herramienta de análisis para el campo de la biología molecular. Tal es el caso en que se encontró (en los casos 1, 2 y 3) que la secuencia hiv2ben no pertenecía al conjunto de secuencias definidas por la máquina de Turing que se obtuvo a través del algoritmo genético. Posteriormente O. Chavoya [Chav94] localizó irregularidades en la codificación al principio de la secuencia hiv2ben (el tramo conocido como long terminal repeat). Así mismo, es posible llevar a cabo la alineación de las secuencias de HIV-2 (en el caso 5); actividad que puede ser muy costosa computacionalmente, si se emplean otras metodologías de alineación (como el método de fuerza bruta, es decir, comparar símbolo por símbolo).

El dominio de problemas que el modelo de generación atomática de máquinas de Turing puede resolver se amplia considerablemente mediante el enfoque de máquina de Turing como computador de funciones. En este caso se propone codificar los argumentos de la función en la cinta de entrada (por medio de un alfabeto unario) y se establece el resultado deseado en la cinta después de la simulación de la máquina. El criterio de mérito puede basarse en la consideración de la distancia de Hamming entre la cinta que resulta de la

simulación de la máquina de Turing y la cinta deseada, para guiar la búsqueda en el espacio de soluciones.

Apéndice A

Implementación Computacional de un Algoritmo Genético.

A continuación se presenta la implementación de un algoritmo genético que ha sido empleado en la solución de diferentes proyectos de investigación en el Área de Cómputo Avanzado del ITESM-CEM [Chav93]. El programa fue codificado en lenguaje C, compilado y ejecutado en una estación de trabajo SUN SparcClassic con el sistema operativo Solaris Versión 2.3.

Como en cualquier programa codificado en lenguaje C, es necesario la inclusión de librerías para la ejecución de diferentes funciones:

```
#include <stdlib.h>
#include <stdio.h>
```

Para la generación de eventos con probabilidad p se puede usar, el macro:

```
#define event(p)  (rand() <= ((p)*((long)RAND_MAX+1)-1))
```

Se utiliza el mismo macro para generar los eventos de recombinación.

```
#define crossover() (event(pc))
```

Se define el macro **random** para la generación de números aleatorios entre 0 y n.

```
#define random(n) ((unsigned char)rand()%(n))
```

Se define el conjunto de parámetros del algoritmo genético de la siguiente manera:

```
unsigned popsize,generations;
unsigned gensize;
float pm=0.01,pc=0.6;
float C=2.0;
```

Donde la variables definidas indican:

popsize. Tamaño de la población.

generations. Número predeterminado de generaciones.

gensize. Tamaño del cromosoma.

pm Probabilidad de mutación.

pc Probabilidad de cruzamiento.

C Constante de escalamiento del fitness.

Las estructuras de datos que permiten la representación de los elementos de la población, son:

```
unsigned char **population;
float    *objective;
float    *fitness;
unsigned char *_son,*_daughter;
unsigned *stack,stackpointer;
unsigned *selectnumber,selectcounter;
float    a,b;
```

population. Apuntador a una secuencia de caracteres (string) que representa a los elementos de la población.

objetive. Apuntador a una secuencia de variables de punto flotante que contiene el valor de la función objetivo asociado al cromosoma.

fitness. Apuntador a una secuencia de variables de punto flotante que contiene el valor del fitness asociado al cromosoma.

_son, _daughter Apuntadores a una secuencia de caracteres que contiene a los elementos de la población generados por medio del operador de recombinación.

stack, stackpointer, selectnumber, selectcounter, a, b
Estructuras para la implementación de un stack y variables que facilitan la selección de los mejores elementos para apareamiento.

La rutina principal **evolve** toma el control del algoritmo genético al evocar a rutinas que llevan a cabo la inicialización del algoritmo, la aplicación de los operadores genéticos a la población, y la finalización del programa.

```
void evolve(void)
        {
        unsigned i,j;
        startevolution();
                for(i=0;i<generations;++i)
                {
                printf("generation %d\n",i);
                evalpopulation();
                mate();
                for(j=0;j<popsize;++j)
                        if(objective[j]==optim_value)
                                break;
                }
        for(i=0;i<popsize;++i)
                objective[i]=eval(population[i]);
        endevolution();
        }
```

La proceso de inicialización es realizado por la rutina **startevolution** en la que principalmente se lleva a cabo la asignación de memoria a las estructuras de datos defindas anteriormente.

```
void startevolution(void)
        {
        unsigned i,j;

        population=(unsigned char **)calloc(popsize,sizeof(unsigned char *));
        objective=(float *)calloc(popsize,sizeof(float));
        fitness=(float *)calloc(popsize,sizeof(float));
        selectnumber=(unsigned *)calloc(popsize,sizeof(unsigned));
        stack=(unsigned *)calloc(popsize,sizeof(unsigned));
        _son=(unsigned char *)calloc(gensize,sizeof(unsigned char));
        _daughter=(unsigned char *)calloc(gensize,sizeof(unsigned char));
        pointer=(unsigned *)calloc(popsize,sizeof(unsigned));
        if(population==NULL || objective==NULL || fitness==NULL ||
                selectnumber==NULL || stack==NULL || _son==NULL
```

```
            || _daughter==NULL || pointer==NULL)
            error(MEMORYERROR,0);
    for(i=0;i<popsize;++i)
            {
            population[i]=(unsigned char *)calloc(gensize,sizeof(unsigned char));
            if(population[i]==NULL)
                    error(MEMORYERROR,1);
            for(j=0,k=0;j<gensize;++j)
                    population[i][j]=random(2);
            }
    }
```

La rutina **evalpopulation** efectúa el proceso de evaluación y selección de los mejores elementos de la población, dando lugar a una población intermedia.

```
void evalpopulation(void)
        {
        unsigned i,j,counter,temporal,score;
        double maxobjective=0,meanobjective=0,globalfitness=0,a,b;
        stackpointer=0;
        selectcounter=popsize;
        for(i=0;i<popsize;++i)
                {
                objective[i]=eval(population[i]);
                printf("%d ",score=objective[i]);
                if(maxobjective<objective[i])
                        maxobjective=objective[i];
                meanobjective+=objective[i];
                }
        meanobjective/=popsize;
        if( (b=(maxobjective-meanobjective)) < (epsilon*maxobjective))
                {
                for(i=0;i<popsize;++i)selectnumber[i]=1;
                }
        else
                {
                a=(C-1)*meanobjective/b;
                b=meanobjective*(maxobjective-C*meanobjective)/b;
                for(i=0;i<popsize;++i)
                        {
                        fitness[i]=a*objective[i]+b;
                        if(fitness[i]<0)fitness[i]=0;
                        globalfitness+=fitness[i];
                        }
                counter=popsize;
                for(i=0;i<popsize;++i)
                        {
                        fitness[i]=fitness[i]*popsize/globalfitness;
                        selectnumber[i]=fitness[i];
                        counter-=selectnumber[i];
                        fitness[i]-=selectnumber[i];
                        }
                if(counter==0)goto exit;
                for(i=0;i<popsize;++i)
                        {
```

```
                pointer[i]=i;
                j=i;
                while(j>0 && fitness[pointer[j]]>fitness[pointer[j-1]])
                        {
                        temporal=pointer[j];
                        pointer[j]=pointer[j-1];
                        pointer[j-1]=temporal;
                        j=j-1;
                        }
                }
        for(i=0;i<counter;++i)
                selectnumber[pointer[i]]+=1;
        }
exit:
for(i=0;i<popsize;++i)
        if(selectnumber[i]==0)
                {
                push(i);
                }
}
```

El proceso de evaluación y selección que realiza **evalpopulation** se favorece con la implementación de un stack. Las rutinas que permiten el uso del stack son **push** y **pop**.

```
void push(unsigned i)
        {
        stack[stackpointer++]=i;
        }

unsigned pop(void)
        {
        return stack[--stackpointer];
        }
```

La rutina **eval** depende del problema y sirve evaluar a los elementos de la población.

```
float eval(unsigned char *gene)
        {
        }
```

El apareamiento de los elementos de la población intermedia es implementado por medio de la rutina **mate** que, de esta manera, genera a la población que conformará a la siguiente generación.

```
void mate(void)
        {
        unsigned i,j,k,mother,father,daughter,son,crosspoint;
        unsigned char *temporal;
```

```
for(i=0;i<popsize;i+=2)
        {
        mother=select();
        father=select();
        if(crossover())
                crosspoint=random(gensize-1)+1;
         else
                crosspoint=gensize;
        son=pop();
        daughter=pop();
        for(j=0,k=0;j<crosspoint;++j)
                {
                _son[j]=copy(population[mother][j],j);
                _daughter[j]=copy(population[father][j],j);
                }
        for(;j<gensize;++j)
                {
                _daughter[j]=copy(population[mother][j],j);
                _son[j]=copy(population[father][j],j);
                }
        temporal=population[son];
        population[son]=_son;
        _son=temporal;
        temporal=population[daughter];
        population[daughter]=_daughter;
        _daughter=temporal;
        }
}
```

La rutina **select** como su nombre lo indica lleva a cabo la selección de parejas para apareamiento en la población intermedia.

```
unsigned select(void)
        {
        unsigned i,index;
        index=random(selectcounter)+1;
        selectcounter--;
        for(i=0;i<popsize;++i)
                {
                if(index<=selectnumber[i])
                        {
                        if((--selectnumber[i])==0)
                                                push(i);
                        return i;
                        }
                index-=selectnumber[i];
                }
        return i;
        }
```

La rutina **copy** es evocada por **mate** para aplicar el operador de mutación mientras lleva a cabo la transcripción de los nuevos elementos de la población.

```
unsigned copy(unsigned c,unsigned k)
        {
        unsigned newc;
        if(event(pm))
                {
                while(1)
                        {
                        newc=random(k);
                        if(newc!=c)return newc;
                        }
                }
        else
                return c;
        }
```

Finalmente, la rutina **endevolution** es empleada para liberar la memoria asignada a las estructuras de datos utilizadas por el programa.

```
void endevolution(void)
        {
        unsigned i;
        for(i=0;i<popsize;++i)
                free(population[i]);
        free(population);
        free(objective);
        free(fitness);
        free(selectnumber);
        free(stack);
        free(_son);
        free(_daughter);
        puts("end of program");
        }
```

Apéndice B

Descripción de programas

1. Counterex

El programa **counterex** genera archivos contra ejemplo del archivo que se le presenta como entrada mediante la aplicación de permutaciones al contenido del mismo. El programa inicialmente lleva a cabo un conteo de las frecuencias de cada uno de los símbolos, que se conservan en el archivo contra ejemplo.

La ejecución se lleva a cabo a través de la siguiente especificación en la línea de comandos:

%counterex <inputfile> <outputfile> <seed> <iteractions>

2. Counterex.prefix

Este programa genera archivos contra ejemplo (de la misma manera que **counterex**) que conservan un prefijo de tamaño especificado correspondiente al archivo de entrada. Se ejecuta de la siguiente manera:

%counterex <inputfile> <outputfile> <seed> <iteractions> <prefixsize>

3. Counterex.stats

Este programa genera un conjunto de archivos contra ejemplo que conservan las propiedades estadísticas de monómeros, dímeros y trímeros correspondientes al conjunto de archivos de entrada. Lo anterior se consigue

mediante el cálculo de las densidades de probabilidad condicional que permiten la construcción sistemática de los archivos contra ejemplo. El programa se ejecuta de la siguiente manera:

%counterex <seed> <#outputs> <#inputs> <input1> ... <inputn>");

4. Reduce

El programa **reduce** lleva a cabo la reducción (a su condición de mínimos estados) de la máquina de Turing obtenida por medio del algoritmo genético. El programa se ejecuta mediante el siguiente comando:

%reduce <inputfile> <outputfile>

5. Sim

El programa **sim** lleva a cabo un postprocesamiento de la salida del sistema **turing** y presenta la simulación de la máquina de Turing mediante la generación de un archivo que contiene las descripciones instantáneas de la máquina operando sobre cada una de las secuencias de entrenamiento. El programa se ejecuta de la siguiente manera:

%sim <inputfile> <outputfile>

Referencias bibliográficas

[Chav93] O. Chavoya, *Apuntes del curso "Algoritmos Genéticos" impartido vía satélite*, Programa de Educación a Distancia, ITESM-CEM, verano 1993.

[Chav94] O. Chavoya, A. Quiroz, M. Quintana *Structural Syntactic Consensus for Studying AIDS Viruses Genetic Diversity*, ITESM-CEM, en refereo en la revista Nature.

[Davi92] Y. Dadivor, H. P. Schwefel, "*An Introduction to Adaptative Optimization Algorithms Based on Principles of Natural Evolution*", en B. Soucek and The IRIS Group, *Dynamic, Genetic, and Chaotic Programming*, John Wiley & Sons, Inc., New York, 1992.

[Denn78] P. J. Denning, J. B. Dennis, and J. E. Qualitz, *Machines, Languages, and Computation*, Prentice Hall, Englewood Cliffs, New Jersey., 1978.

[Forr91] S. Forrest, *Parallelism and Programing in Classifier Systems*, Pittman, London, 1991.

[Fowe66] L. Fowel, A. Owens, y M. Walsh, *Artificial Intelligence Trough Simulated Evolution*, . John Wiley and Sons, Inc., New York, 1966.

[Frie58] R. M. Friedberg, *A Learning Machine: Part I*, IBM Journal of Research and Development, Vol. 2, No. 1, 1958.

[Frie59] R. M. Friedberg, *A Learning Machine: Part II*, IBM Journal of Research and Development, Vol. 3, No. 3, 1959.

[Gold89] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, Mass., 1989.

[Holl75] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[Holl86] J. H. Holland, K. J. Holyoak, R. E. Nisbett, y P.A. Thagard, *Induction: Processes of Inference, Learning, and Duscovery*, MIT Press, Cambridge, Mass., 1986.

[Hopc79] J. E. Hopcroft, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading Mass., 1979.

[Koza92] J. R. Koza, *Genetic Programming*, MIT Press, Cambridge, Mass., 1992.

[Lena76] D. B. Lenat, *"A.M. An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search"*, Ph. D. Dissertation, Computer Science Dept., Stanford University, 1976.

[Lena83] D. B. Lenat, and J. S. Brown, *Why AM and EURISKO appear to work*, Artificial Intelligence, 23, pp. 243-306.

[Mich83] R. S. Michalsky, J. G. Carbonell, y T.M. Mitchell, *Machine Learning: An Artificial Intelligence Approach*, Vol. I, Morgan Kaufman, Los Altos, Calif., 1983.

[Mins67] M. Minsky , *Computation: Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, New Jersey, 1967.

[Ramí94] J. Ramírez, *Estrategias Evolutivas para la Programación Lógica Inductiva*, Tesis de maestría. ITESM - CEM, 1994.

[Rech73] I. Rechenberg, *Evolutionsstrategie: Optimierung Technischer Systeme nach Prizipien der biologischen Evolution*, Frommann-Holzboog Verlag, Stutgart, 1973.

[Schw75] H.P. Schwefel, *Evolutionsstrategie and numerischche Optimierung*, dissertation, Technische Universitat Berline, 1975.

[Sieg91] H.T. Siegelmann , Sontag E. D. "*On the computational power of neural nets*". Rutgers Center for Systems and Control. Noviembre 1991.

[Souc92] B. Soucek and The IRIS Group, *Dynamic, Genetic, and Chaotic Programming*, John Wiley & Sons, Inc., New York, 1992.

[Turi36] A. M. Turing, *On Computable Numbers, with an aplication to the Entscheidungsproblem*, Proc. London Math. Soc, 2-42, pp. 230-245.

[Wils87] S.W. Wilson, *Classifier Systems and the Animat Problem*, Machine Learning, 3, pp. 199-228., 1987.