Instituto Tecnológico de Tuxtla Gutiérrez

Ingeniería Electrónica

Reporte Final

Residencia Profesional:
Visión por Computadora e Inteligencia Artificial aplicado a la ingeniería

Empresa:
Centro de Investigación en Óptica

Presentado por:
Enrique Rubén Molina Pérez

Asesor Interno:
Ing. Alvaro Hernández Sol

Asesor Externo:
Dr. Francisco Javier Cuevas de la Rosa

León, Guanajuato; a 20 de Diciembre de 2013

Índice General

Capítulo 1 ..... - 1 -

Introducción. ..... - 1 -

1.1 Antecedentes. ..... - 2 -

1.2 Justificación. ..... - 4 -

1.3 Objetivo General. ..... - 4 -

1.4 Objetivos Específicos. ..... - 5 -

1.5 Caracterización del área en el que participó. ..... - 5 -

1.6 Problemas a resolver priorizándolos. ..... - 5 -

1.7 Alcances y limitaciones. ..... - 7 -

Capítulo 2 ..... - 8 -

Fundamento Teórico. ..... - 8 -

2.1 Dispositivos de captura de imágenes. ..... - 9 -

2.2 La evolución ..... - 14 -

2.3 Mecanismos de cambio en la evolución. ..... - 16 -

2.4 Evolución de la informática evolutiva. ..... - 18 -

2.5 Introducción a los algoritmos genéticos. ..... - 21 -

2.6 Anatomía de un algoritmo genético. ..... - 22 -

2.7 El zen y los algoritmos genéticos. ..... - 23 -

2.8 Codificación de las variables. ..... - 25 -

Capítulo 3 ..... - 27 -

Desarrollo ..... - 27 -

3.1 Operadores individuales. ..... - 27 -

3.1.1 Operador identidad. ..... - 28 -

3.1.2 Operador inverso o negativo. ..... - 29 -

3.1.3 Operador umbral. ..... - 29 -

3.1.4 Operador intervalo de umbral binario. ..... - 30 -

3.1.5 Operador intervalo de umbral binario invertido. ..... - 31 -

3.1.6 Operador de umbral de la escala de grises. ..... - 31 -

3.1.7 Operador de umbral de la escala de grises invertido. ..... - 32 -

3.1.8 Operador de extensión. ..... - 33 -

3.1.9 Operador reducción del nivel de gris. ..... - 33 -

3.2 Operadores de vecindad. ..... - 34 -

3.3 Modelos de optimización. ..... - 34 -

3.3.1 Cromosoma y función de aptitud. ..... - 36 -

3.3.2 Selección de los mejores individuos. ..... - 40 -

3.3.3 Cruce. ..... - 40 -

3.3.4 Mutación. ..... - 41 -

3.4 Resolución de problemas de optimización con Algoritmos Genéticos. ..... - 47 -

3.5 Resultados, gráficas y programas. ..... - 49 -

3.5.1 Resultados de los modelos de optimización. ..... - 49 -

3.5.2 Código utilizado para la resolución del problema de diseño de la lata. ..... - 72 -

3.6 Conclusiones y recomendaciones. ..... - 77 -

3.7 Referencias Bibliográficas y citas. ..... - 78 -

Índice de figuras

Figura 2.1 Proceso general de la visión por computador. ..... - 9 -
Figura 2.2 Espectro Electromagnético. ..... - 10 -
Figura 2.3 Digitalización de una imagen. ..... - 12 -
Figura 2.4 Macromolécula de ADN. ..... - 15 -
Figura 2.5 Cadena binaria o cromosoma. ..... - 25 -
Figura 3.1 Fotografía de la entrada al Instituto Tecnológico de Tuxtla Gutiérrez. ..... - 28 -
Figura 3. 2 Imagen en escala de grises. ..... - 28 -
Figura 3. 3 Operador inverso aplicado a figura 3.1. ..... - 29 -
Figura 3. 4 Operador umbral aplicado a la figura 3.1. ..... - 30 -
Figura 3. 5 Operador intervalo de umbral binario aplicado a la figura 3.1. ..... - 30 -
Figura 3. 6 Operador intervalo de umbral binario invertido aplicado a la figura 3.1. ..... - 31 -
Figura 3. 7 Operador de umbral de la escala de grises aplicado a la figura 3.1. ..... - 32 -
Figura 3. 8 Operador de umbral de la escala de grises invertido aplicado a la figura 3.1 ..... - 32 -
Figura 3. 9 Operador extensión aplicado a la figura 3.1. ..... - 33 -
Figura 3. 10 Operador reducción del nivel de gris aplicado a la figura 3.1. ..... - 34 -
Figura 3. 11 Diagrama a bloques de un Algoritmo Genético. ..... - 35 -
Figura 3. 12 Menú de funciones en el programa ..... - 36 -
Figura 3. 13 Método menú() ..... - 36 -
Figura 3. 14 Petición de datos para crear la población inicial. ..... - 37 -
Figura 3. 15 Población inicial, decodificación de cromosomas y datos de dispersión. ..... - 37 -
Figura 3. 16 Generación de la población inicial. ..... - 38 -
Figura 3. 17 Algoritmo para la decodificación de cromosomas. ..... - 38 -
Figura 3. 18 Operador de selección con método ruleta. ..... - 40 -
Figura 3. 19 Operador cruce ..... - 41 -
Figura 3. 20 Recombinación de información entre individuos con el operador cruce. ..... - 41 -
Figura 3. 21 Operador mutación. ..... - 42 -
Figura 3. 22 Población final. ..... - 42 -
Figura 3. 23 Gráfica de la función objetivo e individuos de la población inicial. ..... - 43 -
Figura 3. 24 Grafica de la generación 12. ..... - 44 -
Figura 3. 25 Grafica de la generación 24. ..... - 45 -
Figura 3. 26 Gráfica de la generación 36. ..... - 46 -
Figura 3. 27 Gráfica de la generación 48. ..... - 46 -
Figura 3. 28 Población inicial, función objetivo y restricción. ..... - 48 -
Figura 3. 29 Población final, parámetros de la lata optimizados. ..... - 49 -

Capítulo 1

Introducción.

La Inteligencia Artificial es la rama de la ciencia que desarrolla algoritmos entendibles por las máquinas y dispositivos electrónicos con la intención de reproducir las tareas y procesos humanos.

Los algoritmos genéticos (AG), como parte de la Inteligencia Artificial, proporcionan un método de aprendizaje basado en la analogía con la evolución de las especies. Los AG generan un conjunto de hipótesis, llamada normalmente población, mediante la mutación y recombinación de parte del conjunto de hipótesis conocido. En cada paso el conjunto de hipótesis conocido como población actual, se renueva remplazando una proporción de esta población por los sucesores de las hipótesis mejor adaptadas mediante el uso de una función de evaluación.

La popularidad de los AG se debe en parte a que la evolución es un método robusto y bien probado dentro de los sistemas biológicos naturales. Además son fácilmente paralelizables, lo que supone una ventaja gracias al abaratamiento actual de los costes en hardware. Por otra parte, los AG pueden realizar búsquedas en espacios de hipótesis que contienen complejas interacciones entre las distintas partes, donde el impacto de cada parte sobre la función de evaluación es difícil de especificar.

Aunque no se garantice encontrar la solución óptima, los AG generalmente encuentran soluciones con un alto grado de acierto.

La Visión Artificial es parte de la Inteligencia Artificial y se refiere a un conjunto de técnicas y modelos que permiten procesar y analizar la información obtenida a través de una imagen digital. Los Algoritmos Genéticos y la Visión Artificial son ampliamente aplicados en medicina, robótica, arqueología y otros campos que requieren de la toma de decisiones.

1.1 Antecedentes.

Una de las primeras aplicaciones en la mejora de la calidad de imágenes fue el tratamiento de las mismas para su publicación en periódicos enviadas por cable submarino entre Londres y Nueva York.

La introducción del sistema de transmisión de imágenes por cable a través del Atlántico, conocido como sistema Bartlane, a principio de los años 20 redujo el tiempo de transmisión desde más de una semana a menos de tres horas.

Inicialmente, el sistema era capaz de codificar imágenes en 5 niveles de gris distintos, capacidad que se vio aumentada a 15 niveles hacia 1929. Las mejoras en los métodos de transmisión de imágenes continuaron durante los siguientes 35 años.

No obstante, la aparición de los primeros computadores digitales y los primeros programas espaciales en EE. UU. impulsaron los conceptos de procesamiento de imágenes de una forma significativa.

En 1964 se utilizaron técnicas de computador en el Jet Propulsion Laboratory (Pasadena California) para mejorar las imágenes de la Luna enviadas por la sonda espacial Ranger 7.

Estas técnicas sirvieron como base para los métodos de mejora empleados en misiones posteriores, por ejemplo las series Mariner a Marte y Apolo a la Luna. Desde 1964 hasta nuestros días, el campo de procesamiento de imágenes ha crecido enormemente.

Por otra parte, los primeros ejemplos de lo que hoy podríamos llamar algoritmos genéticos aparecieron a finales de los 50 y principios de los 60, programados en computadoras por biólogos evolutivos que buscaban explícitamente realizar modelos de aspectos de la evolución natural. A ninguno de ellos se le ocurrió que esta estrategia podría aplicarse de manera más general a los problemas artificiales, pero ese reconocimiento no tardaría en llegar.

En 1962, investigadores como G.E.P. Box, G.J. Friedman, W.W. Bledsoe y H.J. Bremermann habían desarrollado independientemente algoritmos inspirados en la evolución para optimización de funciones y aprendizaje automático, pero sus trabajos generaron poca reacción.

En 1965 surgió un desarrollo más exitoso, cuando Ingo Rechenberg, entonces de la Universidad Técnica de Berlín, introdujo una técnica que llamó estrategia evolutiva. En esta técnica no había población ni cruzamiento; un padre mutaba para producir un descendiente, y se conservaba el mejor de los dos, convirtiéndose en el padre de la siguiente ronda de mutación (Haupt y Haupt 1998[34], p.146).

Versiones posteriores introdujeron la idea de población. Las estrategias evolutivas todavía se emplean hoy en día por ingenieros y científicos, sobre todo en Alemania.

El siguiente desarrollo importante en el campo vino en 1966, cuando L.J. Fogel, A.J. Owens y M.J. Walsh introdujeron en América una técnica a la que llamaron programación evolutiva. En este método, las soluciones candidatas para los problemas se representaban como máquinas de estado finito sencillas; al igual que en la estrategia evolutiva de Rechenberg, su algoritmo funcionaba mutando aleatoriamente una de estas máquinas simuladas y conservando la mejor de las dos (Mitchell 1996[47], p.2; Goldberg 1989[29], p.105).

También al igual que las estrategias evolutivas, hoy en día existe una formulación más amplia de la técnica de programación evolutiva que todavía es un área de investigación en curso. Sin embargo, lo que todavía faltaba en estas dos metodologías era el reconocimiento de la importancia del cruzamiento.

En una fecha tan temprana como 1962, el trabajo de John Holland sobre sistemas adaptativos estableció las bases para desarrollos posteriores; y lo que es más importante, Holland fue también el primero en proponer explícitamente el cruzamiento y otros operadores de recombinación.

Sin embargo, el trabajo fundamental en el campo de los algoritmos genéticos apareció en 1975, con la publicación del libro “Adaptation in Natural and Artificial Systems”.

Basado en investigaciones anteriores del propio Holland y de colegas de la Universidad de Michigan, este libro fue el primero en presentar sistemática y rigurosamente el concepto de sistemas digitales adaptativos utilizando la mutación, la selección y el cruzamiento, simulando el proceso de la evolución biológica como estrategia para resolver problemas.

El libro también intentó colocar los algoritmos genéticos sobre una base teórica firme introduciendo el concepto de esquema (Mitchell 1996[47], p.3; Haupt y Haupt 1998[34], p.147). Ese mismo año, la importante tesis de Kenneth De Jong estableció el potencial de los AGs demostrando que podían desenvolverse bien en una gran variedad de funciones de prueba, incluyendo paisajes de búsqueda ruidosos, discontinuos y multimodales (Goldberg 1989[29], p.107).

1.2 Justificación.

Los problemas de optimización, en gran mayoría, encuentran solución en la aplicación de las derivadas o métodos numéricos. Pero existen casos en donde éstas herramientas encuentran dificultades para resolverlos, ya sea por una gran cantidad de datos y por la limitada capacidad de computo.

Es aquí donde gana terreno la aplicación de los algoritmos genéticos, pues al estar basados en la teoría evolucionista de Charles Darwin, se simulan evoluciones en las soluciones producidas hasta encontrar la solución correcta o la más cercana a la correcta.

1.3 Objetivo General.

El presente proyecto tiene como objetivo optimizar el uso de materia prima en la elaboración de latas en el Centro de Investigaciones en Óptica, A. C. en León de los Aldama, Guanajuato; implementando las herramientas de los algoritmos genéticos en el proceso de diseño.

1.4 Objetivos Específicos.

- Comprender las operaciones básicas aplicables a imágenes digitales.
- Comprender los conceptos básicos para el desarrollo de algoritmos genéticos.
- Definición de las variables del problema.
- Escritura del algoritmo específico aplicado al problema en un lenguaje de programación.
- Calcular las variables que optimizan el diseño del objeto según las variables del problema.

1.5 Caracterización del área en el que participó.

El área de trabajo en la cual realicé las actividades correspondientes al trabajo de investigación presentadas como proyecto de residencia es el departamento de Óptica del Centro de Investigaciones en Óptica A.C.

El objetivo del Departamento de “Visión computacional e Inteligencia artificial” es realizar la toma de decisiones adecuadas en base a señales de transductores e imágenes digitales.

Los campos donde desarrolla investigación y aplicaciones el departamento de Visión computacional e Inteligencia artificial son: inspección industrial, análisis de imágenes médicas y automatización de procesos.

1.6 Problemas a resolver priorizándolos.

- Implementación de los operadores aplicables a imágenes digitales para la obtención de información.
- Definir las variables involucradas en la optimización del problema.
- Escritura del algoritmo genético aplicable al problema de diseño de la lata.

Cronograma de Actividades.

Actividad	Semana															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Reconocimiento del área de trabajo.	x	x														
Capacitación	x	x	x													
Investigación y documentación.	x	x	x	x	x											
Desarrollo de algoritmos.					x	x	x	x	x							
Pruebas preliminares.									x	x	x	x	x	x		
Documentación del proyecto.							x	x	x	x	x	x	x	x	x	x

Reconocimiento del área de trabajo. En la primera etapa me asignaron un cubículo para realizar el trabajo, así como la presentación de las personas con quienes tendría comunicación, tanto técnica como administrativamente.

Capacitación. En esta etapa me enseñaron a usar el software para la simulación y desarrollo del procesamiento de imágenes y la escritura de los algoritmos genéticos.

Investigación y documentación. En este periodo se llevó a cabo la investigación de los temas que me serían útiles para desarrollar satisfactoriamente mis actividades.

Pruebas preliminares. Se realizaron pruebas preliminares para la detección y depuración de los errores.

Documentación del proyecto. Se escribió un informe global con el fundamento teórico, desarrollos matemáticos, algoritmos computacionales y los resultados obtenidos durante la investigación.

1.7 Alcances y limitaciones.

Alcances

- Tener un control sobre el aprovechamiento de la materia prima.
- Reducción en el costo económico del proceso.

Limitaciones

- Falta de tiempo para el desarrollo completo del prototipo.

Capítulo 2

Fundamento Teórico.

En éste capítulo se tratan los temas que fundamentan este proyecto tales como visión artificial, transformación de imágenes, suavizado, realzado y correcciones radiométricas y algoritmos genéticos.

La visión es uno de los mecanismos sensoriales de percepción más importantes en el ser humano aunque evidentemente no es exclusivo ya que una incapacidad visual no impide en absoluto el desarrollo de ciertas actividades mentales.

El interés de los métodos de procesamiento de imágenes digitales se fundamenta en dos áreas principales de aplicación: a) mejora de la calidad para la interpretación humana; b) procesamiento de los datos de la escena para la percepción de las máquinas de forma autónoma.

En el intento por dotar a las máquinas de un sistema de visión aparece el concepto de Visión Artificial. En la mayoría de los sistemas la capacidad sensorial de visión es complementada con otros mecanismos sensoriales tales como detectores de alcance o proximidad [1]. Posteriormente es necesario realizar una integración multisensorial para completar el proceso de percepción global.

Las técnicas de procesamiento se usan ahora para resolver una gran variedad de problemas. Aunque a menudo no relacionados, esos problemas requieren comúnmente métodos capaces de realzar y extraer la información contenida en las imágenes para su interpretación y análisis por parte de los humanos. En cualquier caso se contemplan tanto técnicas de mejora de la calidad de las imágenes como relativas a la percepción de máquina.

Desde una percepción general, la visión artificial por computador es la capacidad de la máquina para ver el mundo que le rodea, más precisamente para deducir la estructura y las propiedades del mundo tridimensional a partir de una o más imágenes bidimensionales.

El proceso general se puede sintetizar en la figura 2.1, en la que las cajas representan datos y las burbujas procesos. Se parte de la escena tridimensional y se termina con la aplicación de interés.

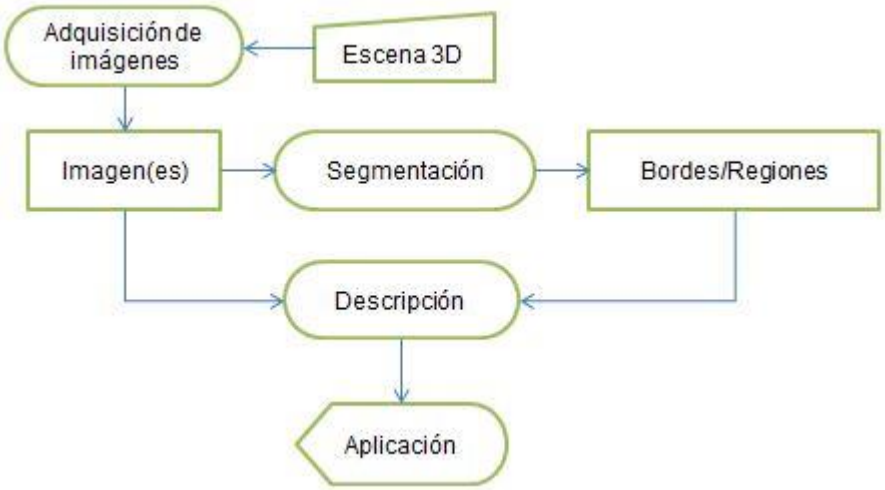


Figura 2.1 Proceso general de la visión por computador.

En la visión por computador, la escena tridimensional es vista por una, dos o más cámaras para producir imágenes monocromáticas o en color. Las imágenes adquiridas pueden ser segmentadas para obtener de ellas características de interés tales como bordes o regiones. Posteriormente, de las características se obtienen las propiedades subyacentes mediante el correspondiente proceso de descripción. Tras lo cual se consigue la estructura de la escena tridimensional requerida por la aplicación de interés.

Las imágenes de intensidad están íntimamente ligadas al concepto de luminosidad.

2.1 Dispositivos de captura de imágenes.

Para la adquisición de imágenes digitales se requieren dos elementos básicos. El primero es un dispositivo físico que es sensible a una determinada banda del espectro de energía electromagnético (tal como rayos X, ultravioleta, visible, infrarrojos, etc.) y que produce una señal eléctrica de salida proporcional al nivel de energía incidente en cualquier instante de tiempo.

En la figura 2.2 se muestra el rango de frecuencias del espectro electromagnético.

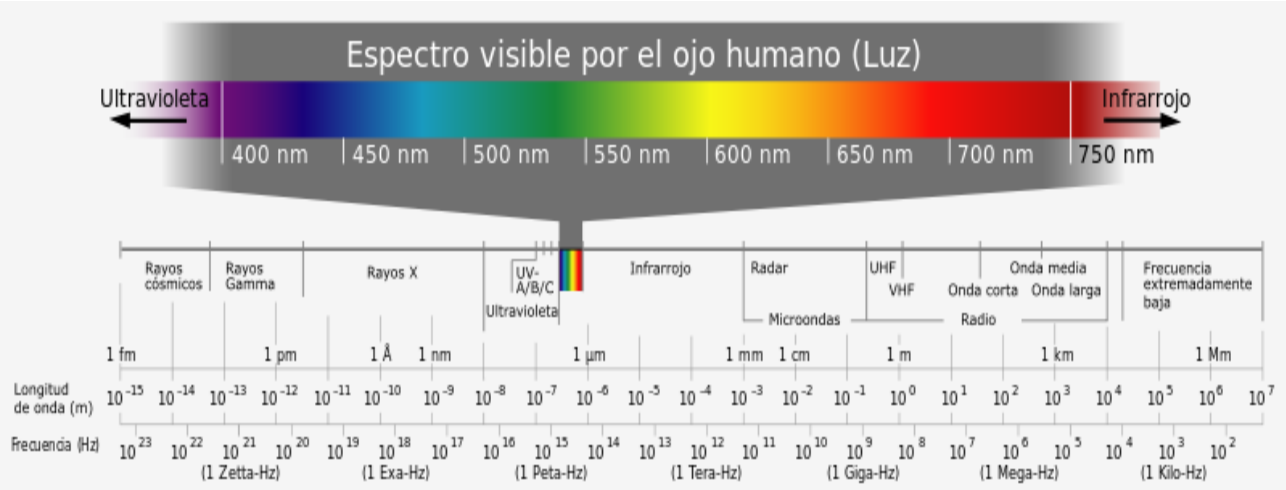


Figura 2.2 Espectro Electromagnético.

El segundo, denominado digitalizador, es un dispositivo para convertir una señal por ejemplo eléctrica continua de salida del dispositivo físico en un conjunto discreto de localizaciones del plano de la imagen y, después, en la cuantización de dicha muestra. Esto implica, en primer lugar, determinar el valor de la imagen continua en cada una de las localizaciones discretas de la imagen (cuyos valores se denominan muestras de la imagen) y en segundo lugar asignar a cada muestra una etiqueta entera discreta, representativa del rango de valores en el que varía la muestra. Una vez cuantificadas esas señales espacialmente y en amplitud se obtiene una imagen digital, que es como se representa en el computador. Es decir, tendremos una matriz de números enteros en la cual cada valor entero representa la intensidad de los objetos de la escena en un tiempo discreto y en un punto discreto del plano de la imagen.

Los dispositivos de captura de imágenes principalmente usados para la visión artificial son las cámaras de televisión y los dispositivos de acoplamiento de carga (Charge Coupled Devices –CCD-). Los primeros producen una señal continua de video de suerte que el número de líneas propio de cada sistema de vídeo y la frecuencia de muestreo originan lo que se conoce como nivel de resolución espacial, el muestreo espacial junto con la cuantización en amplitud origina la mencionada imagen digital. En

cuanto a los dispositivos CCD, que generalmente también producen una señal continua de vídeo, cabe distinguir dos categorías [1]: sensores de exploración de línea y de área.

Un sensor CCD de exploración de línea consta de una fila de M elementos semiconductores llamados *photosites.* Los fotones procedentes de la escena excitan el elemento semiconductor, de forma que el grado de excitación es proporcional a la cantidad de carga acumulada en el photosite y por tanto a la intensidad luminosa o reflectancia en ese punto, Una imagen en este tipo de sensores se obtiene cuando se han captado N imágenes lineales. Este tipo de sensores tienen gran aceptación en aplicaciones en las que los objetos se mueven perpendicularmente a la dirección de captura del sensor, de suerte que dicho movimiento produce la imagen bidimensional. Los sensores de área son similares a los de exploración de línea, pero con la diferencia de que los photosites se agrupan en forma de matriz con M elementos por fila y N elementos por columnas. Son también de uso frecuente los escáneres o los video reproductores.

En definitiva, independientemente del tipo de sensor utilizado, la imagen que ha de ser tratada por el computador se presenta digitalizada espacialmente en forma de matriz con una resolución de *M* x *N* elementos. Cada elemento de la matriz denominado *píxel (picture element)* tendrá un valor asignado, que corresponde con el nivel de luminosidad del punto correspondiente en la escena captada, dicho valor es el resultado de la cuantización de intensidad o nivel de gris. Los términos intensidad y nivel de gris se suelen usar indistintamente. En la figura 2.3 se muestra el proceso de digitalización. El paso final consiste en enviar las imágenes obtenidas a una computadora para su tratamiento mediante programas específicos o desarrollados por el usuario. Las imágenes se pueden almacenar por cualquier medio magnético (en el disco duro, en CD, disquetes, etc.) y pueden ser visualizadas a través de los monitores de TV o del PC, impresoras, etc.

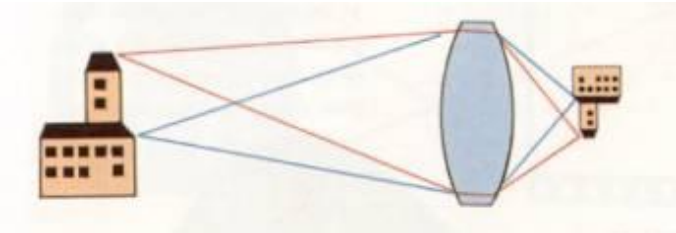


Figura 2.3 Digitalización de una imagen.

En relación con el proceso de captura esquematizado en la figura 2.3, es preciso realizar dos importantes matizaciones:

a) El elemento que aparece como lente en realidad se refiere al sistema óptico necesario en los dispositivos de captura. La óptica del sistema genera ciertas aberraciones bien conocidas en los dispositivos ópticos de los que se ocupa la óptica como rama de la ciencia; también su presencia origina que un punto nítido espacial no se proyecte exactamente con la nitidez esperada, antes bien lo haga con una cierta dispersión alrededor del teórico punto. Es lo que se conoce como dispersión puntual. Por su importancia, esta dispersión conviene modelarla a través de la denominada función de dispersión puntual (*Point Spread Funtion – PSF-*), lo cual no siempre es posible, en cuyo caso se asume que suele corresponderse con una distribución dada (por ejemplo gaussiana) de forma que desde el punto de vista del filtrado de la señal ésta distribución se correspondería con un núcleo de filtro paso bajo.

b) La intensidad que se obtiene finalmente como determinante de cada punto espacial es en realidad el producto de una componente de iluminación y una componente de reflectancia. La primera es consecuencia de las fuentes de iluminación existentes en la escena en el momento de la captura. La segunda está asociada a las propiedades intrínsecas del objeto. Desde el punto de vista de la teoría de la señal la iluminación se corresponde con las bajas frecuencias de la imagen mientras que la reflectancia está relacionada con las altas frecuencias. Existen diferentes mecanismos para separar las componentes de iluminación y reflectancia. Entre ellos destaca el filtrado homomórfico, cuyo

fundamento consiste en eliminar las bajas frecuencias de la imagen para quedarse sólo con las altas y por tanto con los detalles. Esto es útil en aplicaciones donde la iluminación sea muy variable y lo que interese sea evitar esa alta variabilidad. Por ejemplo, la detección de cambios entre imágenes de exterior tomadas bajo diferentes condiciones climáticas donde la iluminación es realmente distinta nos indicaría que toda la imagen ha cambiado si no eliminamos la componente de iluminación.

Los algoritmos genéticos tienen sus antecedentes en la biología y comienzan con Darwin, que con su libro *El origen de las especies por medio de la selección natural o la preservación de las razas favorecidas en su lucha por la vida*, nos habla sobre los principios de la selección natural.

Los principios básicos de los algoritmos genéticos se derivan de *Las leyes de la vida natural* descritos por Darwin [2]:

- Existe una población de individuos con diferentes propiedades y habilidades. Así mismo existe una limitación sobre el número de individuos que existen en una determinada población.
- La naturaleza crea nuevos individuos con propiedades similares a los individuos existentes.
- Los individuos más prometedores se seleccionan más a menudo para la reproducción de acuerdo con la selección natural.

Los algoritmos genéticos imitan los principios de la vida descritos y los utilizan para propósitos de optimización.

Una de las principales deficiencias del argumento de Darwin es que, a pesar de que la herencia juega un papel preponderante en su teoría, no ofrece una explicación acerca de su funcionamiento. Sin embargo, desde Mendel se conoce que la herencia se produce a través del código genético presente en las células reproductivas [3].

2.2 La evolución

Antes de ver el tipo de problemas a los que se pueden aplicar los algoritmos genéticos, se va a estudiar qué es lo que inspira dichos algoritmos, la Naturaleza, ese fenómeno natural denominado evolución.

La teoría de la evolución fue descrita por Charles Darwin 20 años después de su viaje por las islas Galápagos en el Beagle, en el libro *Sobre el origen de las especies por medio de la Selección Natural*. Este libro fue bastante polémico en su tiempo, y en cualquier caso es una descripción incompleta de la evolución. La hipótesis de Darwin, presentada junto con Wallace, que llegó a las mismas conclusiones independientemente, es que pequeños cambios heredables en los seres vivos y la selección son los dos hechos que provocan el cambio en la Naturaleza y la generación de nuevas especies. Pero Darwin desconocía cual es la base de la herencia, pensaba que los rasgos de un ser vivo eran como un fluido, y que los “fluidos” de los dos padres se mezclaban en la descendencia; esta hipótesis tenía el problema de que al cabo de cierto tiempo, una población tendría los mismos rasgos intermedios [9].

Fue Mendel quien descubrió que los caracteres se heredaban de forma discreta, y que se tomaban del padre o de la madre, dependiendo de su carácter dominante o recesivo. A estos caracteres que podían tomar diferentes valores se les llamaron genes, y a los valores que podían tomar, alelos.

En la realidad, las teorías de Mendel, que trabajó en total aislamiento, se olvidaron y no se volvieron a redescubrir hasta principios del siglo XX. Además, hasta 1930 el genetista inglés Robert Aylmer no relacionó ambas teorías, demostrando que los genes mendelianos eran los que proporcionaban el mecanismo necesario para la evolución [3].

Más o menos por la misma época, el biólogo alemán Walther Flemming describió los cromosomas, como ciertos filamentos en los que se agregaban la cromatina del núcleo celular durante la división; poco más adelante se descubrió que las células de cada especie tenían un número fijo y característico de cromosomas.

Fue hasta los años 50, cuando Watson y Crick descubrieron que la base molecular de los genes está en el ADN, ácido desoxirribonucleico, figura 2.4. Los cromosomas están compuestos de ADN, y por tanto los genes están en los cromosomas. La macromolécula de ADN está compuesta por bases. La combinación y la secuencia de éstas bases forma el código genético, único para cada ser vivo.

Figura 2.4 Macromolécula de ADN.

Todos estos hechos forman hoy en día la teoría del neo-darwinismo, que afirma que la historia de la mayoría de la vida está causada por una serie de procesos que actúan en y dentro de las poblaciones: *reproducción, mutación y selección*. La evolución se puede definir entonces como cambios en el área genética de una población.

Un tema polémico, con opiniones variadas dependiendo de si se trata de informáticos evolutivos o de biólogos o genetistas, es si la evolución optimiza o no. Según los informáticos evolutivos, la evolución optimiza, puesto que va creando seres cada vez más perfectos, cuya culminación es el hombre.

Sin embargo, los genetistas y biólogos evolutivos afirman que la evolución no optimiza, sino que adapta y optimiza localmente en el espacio y el tiempo, un organismo más evolucionado puede estar en desventaja competitiva con uno de sus ancestros, si se colocan en el ambiente del último; es decir, la evolución no significa progreso [4].

2.3 Mecanismos de cambio en la evolución.

Estos mecanismos de cambio serán necesarios para entender los algoritmos evolutivos, pues se trata de imitarlos para resolver problemas de ingeniería; por eso merece la atención conocerlos en más profundidad.

Los mecanismos de cambio alteran la proporción de alelos de un tipo determinado en una población, y se dividen en dos tipos: *los que disminuyen la variabilidad* y *los que la aumentan.*

Los principales mecanismos que disminuyen la variabilidad son los siguientes:

- *Selección natural:* los individuos que tengan algún rasgo que los haga menos válidos para realizar su tarea de seres vivos, no llegarán a reproducirse, y por tanto, su patrimonio genético desaparecerá de la población; algunos no llegarán ni siquiera a nacer. Esta selección sucede a muchos niveles: *competición entre miembros de la especie (intraespecífica), competición entre diferentes especies, y competición predador-presa.* También es importante la selección sexual, en la cual las hembras eligen el mejor individuo de su especie disponible para reproducirse.
- *Deriva genética:* el simple hecho de que un alelo sea más común en la población que otro, causará que la proporción de alelos de esa población vaya aumentando en una población aislada, lo cual a veces da lugar a fenómenos de especiación, por ejemplo, por el denominado efecto fundador.

Otros mecanismos aumentan la diversidad, y suceden generalmente en el ámbito molecular. Los más importantes son:

- *Mutación:* la mutación es una alteración del código genético, que puede suceder por múltiples razones. En muchos casos, las mutaciones, que cambian un nucleótido por otro, son letales, y los individuos ni siquiera llegan a desarrollarse, pero a veces se da lugar a la producción de una proteína que aumenta la supervivencia del individuo, y que, por tanto, es pasada a la descendencia. Las mutaciones son totalmente aleatorias, y son el mecanismo básico de generación

de variedad genética. A pesar de lo que se piensa habitualmente, la mayoría de las mutaciones ocurren de forma natural, aunque existen sustancias mutagénicas que aumentan su frecuencia.

- *Poliploidía:* mientras que las células normales poseen dos copias de cada cromosoma, y las células reproductivas una (haploides), puede suceder por accidente que alguna célula reproductiva tenga dos copias; si se logra combinar con otra célula diploide o haploide dará lugar a un ser vivo con varias copias de cada cromosoma. La mayoría de las veces, la poliploidía da lugar a individuos con algún defecto (por ejemplo, el tener 3 copias del cromosoma 21 da lugar al mongolismo), pero en algunos casos se crean individuos viables.
- *Recombinación:* cuando las dos células sexuales, o gametos, una masculina y otra femenina se combinan, los cromosomas de cada una también lo hacen, intercambiándose genes, que a partir de ese momento pertenecerán a un cromosoma diferente. A veces también se produce traslocación dentro de un cromosoma; una secuencia de código se elimina de un sitio y aparece en otro sitio del cromosoma, o en otro cromosoma.
- *Flujo genético:* o intercambio de material genético entre seres vivos de diferentes especies. Normalmente se produce a través de un vector, que suele ser virus o bacterias; estas incorporan a su material genético genes procedentes de una especie a la que han infectado, y cuando infectan a un individuo de otra especie pueden transmitirle esos genes a los tejidos generativos de gametos.

En resumen, la selección natural actúa sobre el fenotipo y suele disminuir la diversidad, haciendo que sobrevivan solo los individuos más aptos, los mecanismos que generan diversidad y que combinan características actúan habitualmente sobre el genotipo [4].

2.4 Evolución de la informática evolutiva.

Ahora que se han descrito cuales son los mecanismos de la evolución, y una amplia gama de problemas que pueden o no tener relación entre sí, llega la hora de contar, desde sus principios, como evolucionó la idea de simular o imitar la evolución con el objetivo de resolver problemas humanos.

Las primeras ideas, incluso antes del descubrimiento del ADN, vinieron de Von Neumann, uno de los mayores científicos de este siglo. Von Neumann afirmó que la vida debía de estar apoyada por un código que a la vez describiera como se puede construir un ser vivo, y tal que ese ser creado fuera capaz de auto reproducirse; por tanto, un autómata o máquina auto reproductiva tendría que ser capaz, aparte de contener las instrucciones para hacerlo, de ser capaz de copiar tales instrucciones a su descendencia [10].

Sin embargo, no fue hasta mediados de los años cincuenta, cuando el rompecabezas de la evolución se había prácticamente completado, cuando *Box* comenzó a pensar en imitarla, para en su caso, mejorar procesos industriales. La técnica de *Box,* denominada EVOP (*Evolutionary Operation*), consistía en elegir una serie de variables que regían un proceso industrial. Sobre esas variables se creaban pequeñas variaciones que formaban un hipercubo, variando el valor de las variables una cantidad fija. Se probaba entonces con cada una de las esquinas del hipoercubo durante un tiempo, y al final del periodo de pruebas, un comité humano decidía sobre la cantidad del resultado. Es decir, se estaba aplicando mutación y selección a los valores de las variables, con el objeto de mejorar la calidad del proceso. Este procedimiento se aplicó con éxito a algunas industrias químicas.

Un poco más adelante, en 1958, Freidberg y sus colaboradores pensaron en mejorar usando técnicas evolutivas la operación de un programa. Para ello diseñaron un código máquina de 14 bits (2 para el código de operación, y 6 para los datos y/o instrucciones); cada programa, tenía 64 instrucciones. Un programa llamado Herman, ejecutaba los programas creados, y otro programa, el Teacher o profesor, le mandaba a Herman ejecutar otros programas y ver si los programas ejecutados habían realizado su tarea o

no. La tarea consistía en leer unas entradas, situadas en una posición de memoria, y debían depositar el resultado en otra posición de memoria, que era examinada al terminarse de ejecutar la última instrucción.

Para hacer evolucionar los programas, Friedberg hizo que en cada posición de memoria hubiera dos alternativas; para cambiar un programa, alternaba las dos instrucciones (que eran una especie de alelos), o bien reemplazaba una de las dos instrucciones con una totalmente aleatoria.

En realidad, lo que estaba haciendo es usar mutación para generar nuevos programas; al parecer, no tuvo más éxito que si hubiera buscado aleatoriamente un programa que hiciera la misma tarea. El problema es que la mutación sola, sin ayuda de la selección, hace que la búsqueda sea prácticamente una búsqueda aleatoria.

Más o menos simultáneamente, Bremmerman trató de usar la evolución para “entender los procesos de pensamiento creativo y aprendizaje”, y empezó a considerar la evolución como un proceso de aprendizaje. Para resolver un problema, codificaba las variables del problema en una cadena binaria de 0s y 1s, y sometía la cadena a mutación, cambiando un bit de cada vez; de esta forma, estableció que la tasa ideal de mutación debía de ser tal que se cambiara un bit cada vez. Bremmerman trató de resolver problemas de minimización de funciones, aunque no está muy claro qué tipo de selección usó, si es que usó alguna, y el tamaño y tipo de la población. En todo caso, se llegaba a un punto, la “trampa de Bremmerman”, en el cual la solución no mejoraba; en intentos sucesivos trató de añadir entrecruzamiento entre soluciones, pero tampoco obtuvo buenos resultados. Una vez más, el simple uso de operadores que creen diversidad no es suficiente para dirigir la búsqueda genética hacia la solución correcta; y corresponde a un concepto de la evolución darwiniano clásico, por mutación, se puede mejorar a un individuo; en realidad, la evolución actúa a nivel de la población.

El primer uso de procedimientos evolutivos en inteligencia artificial se deba a Reed, Toombs y Baricelli, que trataron de hacer evolucionar un tahúr que jugaba a un juego de cartas simplificado. Las estrategias de juego consistían en una serie de 4 probabilidades de apuesta alta o baja con una mano alta o baja, con cuatro parámetros

de mutación asociados. Se mantenía una población de 50 individuos, y aparte de la mutación, había intercambio de probabilidades entre dos padres. Es de suponer que los perdedores se eliminaban de la población (tirándolos por la borda). Aparte de, probablemente, crear buenas estrategias, llegaron a la conclusión de que el entrecruzamiento no aportaba mucho a la búsqueda.

Los intentos posteriores, ya realizados en los años 60, ya corresponden a los algoritmos evolutivos modernos, y se han seguido investigando hasta nuestros días. Algunos de ellos son simultáneos a los algoritmos genéticos, pero se desarrollaron sin conocimiento unos de otros. Uno de ellos, la programación evolutiva de Fogel, se inició como un intento de usar la evolución para crear máquinas inteligentes, que pudieran prever su entorno y reaccionar adecuadamente a él. Para simular una máquina pensante, se utilizó un autómata celular.

Fogel trataba de hacer aprender a estos autómatas a encontrar regularidades en los símbolos que se le iban enviando. Como método de aprendizaje, usó un algoritmo evolutivo: una población de diferentes autómatas competía para hallar la mejor solución, es decir, predecir cuál iba a ser el siguiente símbolo de la secuencia con un mínimo de errores; los peores 50% eran eliminados cada generación, y sustituidos por otros autómatas resultantes de una mutación de los existentes.

De ésta forma, se lograron hacer evolucionar autómatas que predecían algunos números primos (por ejemplo, uno, cuando se le daban los números más altos, respondía siempre que no era primo; la mayoría de los números mayores de 100 son no primos). En cualquier caso, estos primeros experimentos demostraron el potencial de la evolución como método de búsqueda de soluciones novedosas.

Más o menos a mediados de los años 60, Rechenberg y Schwefel describieron las estrategias de evolución. Las estrategias de evolución son métodos de optimización paramétricos, que trabajan sobre poblaciones de cromosomas compuestos por números reales. Hay diversos tipos de estrategias de evolución, que se verán más adelante, pero en la más común, se crean nuevos individuos de la población añadiendo un vector mutación a los cromosomas existentes en la población; en cada generación,

se elimina un porcentaje de la población (especificado por los parámetros y μ), y los restantes generan la población total, mediante mutación y crossover. La magnitud del vector de mutación se calcula adaptativamente [5].

2.5 Introducción a los algoritmos genéticos.

John Holland desde pequeño, se preguntaba cómo logra la naturaleza, crear seres cada vez más perfectos (aunque, como se ha visto, esto no es totalmente cierto, o en todo caso depende de qué entienda uno por perfecto).

Lo curioso era que todo se lleva a cabo a base de interacciones locales entre individuos, y entre estos y lo que les rodea. No sabía la respuesta, pero tenía una cierta idea de cómo hallarla: tratando de hacer pequeños modelos de la naturaleza, que tuvieran alguna de sus características, y ver cómo funcionaban, para luego extrapolar sus conclusiones a la totalidad [6].

De hecho, ya de pequeño hacía simulaciones de batallas célebres con todos sus elementos: copiaba mapas y los cubría luego de pequeños ejércitos que se enfrentaban entre sí.

En los años 50 entró en contacto con los primeros ordenadores, donde pudo llevar a cabo algunas de sus ideas, aunque no se encontró con un ambiente intelectual fértil para propagarlas. Fue a principios de los 60, en la Universidad de Michigan en Ann Arbor, donde, dentro del grupo Logic of Computers, sus ideas comenzaron a desarrollarse y a dar frutos. Y fue, además, leyendo un libro escrito por un biólogo evolucionista, R. A. Fisher, titulado La teoría genética de la selección natural, como comenzó a descubrir los medios de llevar a cabo sus propósitos de comprensión de la naturaleza. De ese libro aprendió que la evolución era una forma de adaptación más potente que el simple aprendizaje, y tomó la decisión de aplicar estas ideas para desarrollar programas bien adaptados para un fin determinado.

En esa universidad, Holland impartía un curso titulado Teoría de sistemas adaptativos. Dentro de este curso, y con una participación activa por parte de sus estudiantes, fue donde se crearon las ideas que más tarde se convertirían en los algoritmos genéticos.

Cuando Holland se enfrentó a los algoritmos genéticos, los objetivos de su investigación fueron dos: imitar los procesos adaptativos de los sistemas naturales, y diseñar sistemas artificiales (normalmente programas) que retengan los mecanismos importantes de los sistemas naturales.

Unos 15 años más adelante, David Goldberg, actual delfín de los algoritmos genéticos, conoció a Holland, y se convirtió en su estudiante. Golberg era un ingeniero industrial trabajando en diseño de pipelines, y fue uno de los primeros que trató de aplicar los algoritmos genéticos a problemas industriales. Aunque Holland trató de disuadirle, porque pensaba que el problema era excesivamente complicado como para aplicarle algoritmos genéticos, Goldberg consiguió lo que quería, escribiendo un algoritmo genético en un ordenador personal Apple II. Estas y otras aplicaciones creadas por estudiantes de Holland convirtieron a los algoritmos genéticos en un campo con base suficientemente aceptado para celebrar la primera conferencia en 1985, ICGA´85. Tal conferencia se sigue celebrando bianualmente [4].

2.6 Anatomía de un algoritmo genético.

Los algoritmos genéticos son métodos sistemáticos para la resolución de problemas de búsqueda y optimización que aplican a estos los mismos métodos de la evolución biológica: selección basada en la población, reproducción sexual y mutación.

Los algoritmos genéticos son métodos de optimización, que tratan de resolver el mismo conjunto de problemas que se ha contemplado anteriormente, es decir, hallar $(x_i,...,x_n)$ tales que $F(x_i,...,x_n)$ sea máximo. En un algoritmo genético, tras parametrizar el problema en una serie de variables, $(x_i,...,x_n)$ se codifican en un cromosoma. Todos los operadores utilizados por un algoritmo genético se aplicarán sobre estos cromosomas, o sobre poblaciones de ellos.

En el algoritmo genético va implícito el método para resolver el problema; son solo parámetros de tal método los que están codificados, a diferencia de otros algoritmos evolutivos como la programación genética. Hay que tener en cuenta que un algoritmo genético es independiente del problema, lo cual lo hace un algoritmo robusto, por ser útil para cualquier problema, pero a la vez débil, pues no está especializado en ninguno.

Las soluciones codificadas en un cromosoma compiten para ver cuál constituye la mejor solución (aunque no necesariamente la mejor de todas las soluciones posibles). El ambiente, constituido por otras soluciones, ejercerá una presión selectiva sobre la población, de forma que sólo los mejor adaptados (aquellos que resuelvan mejor el problema) sobrevivan o leguen su material genético a las siguientes generaciones, igual que en la evolución de las especies. La diversidad genética se introduce mediante mutaciones y reproducción sexual.

En la Naturaleza lo único que hay que optimizar es la supervivencia, y eso significa a su vez maximizar diversos factores y minimizar otros. Un algoritmo genético, sin embargo, se usará habitualmente para optimizar sólo una función, no diversas funciones relacionadas entre sí simultáneamente. La optimización que busca diferentes objetivos simultáneamente, denominada multimodal o multiobjetivo, también se suele abordar con un algoritmo genético especializado.

Por lo tanto, un algoritmo genético consiste en lo siguiente: hallar de qué parámetros depende el problema, codificarlos en un cromosoma, y se aplican los métodos de la evolución: selección y reproducción sexual con intercambio de información y alteraciones que generan diversidad. En las siguientes secciones se verán cada uno de los aspectos del algoritmo genético [4].

2.7 El zen y los algoritmos genéticos.

Este es el título de un artículo que publicó Goldberg en la conferencia sobre algoritmos genéticos celebrada en el año 89 (ICGA 89), en donde da una serie de consejos para

que se apliquen los algoritmos genéticos debidamente, y avisa a aquellos que se quieren apartar de la ortodoxia. Estos consejos son los siguientes:

- *Deja que la Naturaleza sea tu guía:* dado que la mayoría de los problemas a los que se van a aplicar los algoritmos genéticos son de naturaleza no lineal, es mejor actuar como lo hace la naturaleza, aunque intuitivamente pueda parecer la forma menos acertada. Si queremos desarrollar sistemas no lineales que busquen y aprendan, mejor que comencemos (como mínimo) imitando a sistemas que funcionan [7]. Y estos sistemas se hallan en la naturaleza.
- *Cuidado con el asalto frontal:* a veces se plantea el problema de pérdida de diversidad genética en una población de cromosomas. Hay dos formas de resolver este problema: aumentar el ritmo de mutación, lo cual equivale a convertir un algoritmo genético en un algoritmo de búsqueda aleatoria, o bien introducir mecanismos como el sharing, por el cual el fitness de un individuo se divide por el número de individuos similares a él. Este segundo método, más parecido al funcionamiento de la naturaleza, en la cual cada individuo, por bueno que sea, tiene que compartir recursos con aquellos que hayan resuelto el problema de la misma forma, funciona mucho mejor. Otro caso que surge a menudo en los grupos de discusión de Usenet es el tratar de optimizar AGs mediante AGs; es mucho mejor tratar de entender el problema que acercarse a él de esta manera.
- *Respeta la criba de esquemas:* para ello, lo ideal es utilizar alfabetos con baja cardinalidad (es decir, con pocas letras) como el binario.
- *No te fíes de la autoridad central:* la Naturaleza actúa de forma distribuida, por tanto, se debe de minimizar la necesidad de operadores que "vean" a toda la población. Ello permite, además, una fácil paralelización del algoritmo genético. Por ejemplo, en vez de comparar el fitness de un individuo con todos los demás, se puede comparar sólo con los vecinos, es decir, aquellos que estén, de alguna forma, situados cerca de él [4].

2.8 Codificación de las variables.

Los algoritmos genéticos requieren que el conjunto se codifique en un cromosoma. Cada cromosoma tiene varios genes, que corresponden a los parámetros del problema. Para poder trabajar con estos genes en el ordenador, es necesario codificarlos en una cadena.

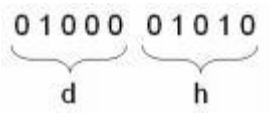


Figura 2.5 Cadena binaria o cromosoma.

Por ejemplo, en esta cadena de bits de la figura 2.5, el valor del parámetro h ocupará las posiciones 0 a 4 y el parámetro d, las posiciones 5 a 9. El número de bits usado para cada parámetro dependerá de la precisión que se quiera en el mismo o del número de opciones posibles (alelos) que tenga ese parámetro.

Hay otras codificaciones posibles, usando alfabetos de diferente cardinalidad; sin embargo, uno de los resultados fundamentales en la teoría de algoritmos genéticos, el teorema de los esquemas, afirma que la codificación óptima es aquella que tiene un alfabeto de cardinalidad 2.

La mayoría de las veces, una codificación correcta es la clave de una buena resolución del problema. Generalmente, la regla heurística que se utiliza es la llamada regla de los bloques de construcción, es decir, parámetros relacionados entre sí deben de estar cercanos en el cromosoma. Por ejemplo, si queremos codificar los pesos de una red neuronal, una buena elección será poner juntos todos los pesos que salgan de la misma neurona de la capa oculta. En todo caso, se puede ser bastante creativo con la codificación del problema, teniendo siempre en cuenta la regla anterior. Esto puede llevar a usar cromosomas bidimensionales, o tridimensionales, o con relaciones entre genes que no sean puramente lineales de vecindad. En algunos casos, cuando no se conoce de antemano el número de variables del problema, caben dos opciones:

codificar también el número de variables, fijando un número máximo, o bien, lo cual es mucho más natural, crear un cromosoma que pueda variar de longitud. Para ello, claro está, se necesitan operadores genéticos que alteren la longitud.

Normalmente, la codificación es estática, pero en casos de optimización numérica, el número de bits dedicados a codificar un parámetro puede variar, o incluso lo que representen los bits dedicados a codificar cada parámetro. Algunos paquetes de algoritmos genéticos adaptan automáticamente la codificación según van convergiendo los bits menos significativos de una solución.

Capítulo 3

Desarrollo

Operadores con imágenes.

Por transformación de una imagen se entiende al proceso de modificar el contenido de una imagen original para obtener una nueva. El objetivo de cualquier transformación estriba en la necesidad de preparar la imagen con el fin de realizar un posterior análisis de cara a su interpretación. La interpretación entra dentro de un proceso de percepción de nivel superior que debe estar implícito en toda aplicación de visión artificial [8].

Para ésta actividad fue necesario utilizar como apoyo el software MathCAD en su versión 14.0. Este programa nos permite el cálculo de operaciones matemáticas y procesamiento de imágenes de manera fácil.

3.1 Operadores individuales.

Las operaciones individuales implican la generación de una nueva imagen modificando el valor del píxel en una simple localización basándose en una regla global aplicada a cada localización de la imagen original. El proceso consiste en obtener el valor del píxel de una localización dada en la imagen, modificándolo por una operación lineal o no lineal y colocando el valor del nuevo píxel en la correspondiente localización de la nueva imagen. El proceso se repite para todas y cada una de las localizaciones de los píxeles en la imagen original.

Para comenzar a trabajar necesitamos importar la imagen con la que queremos trabajar, figura 3.1, en este caso escogimos:

Figura 3.1 Fotografía de la entrada al Instituto Tecnológico de Tuxtla Gutiérrez.

esto lo hacemos con el comando:

M:= READBMP("D:/Recursos/ittg")

La función de éste comando es cargar en la variable *M* los valores de intensidad (escala de grises) de cada píxel de la imagen, entre las comillas escribimos la dirección del archivo en formato .bmp. El resultado es:

Figura 3. 1 Imagen en escala de grises.

3.1.1 Operador identidad.

Este operador crea una imagen de salida que es idéntica a la imagen de entrada. Su función es:

$$q = p \quad (1)$$

Donde p = intensidad del píxel en la imagen original,
q = intensidad del píxel en la imagen nueva.

3.1.2 Operador inverso o negativo.

Este operador crea una imagen de salida que es la inversa de la imagen de entrada. Este operador es útil en diversas aplicaciones tales como imágenes médicas. Para una imagen con valores de gris en el rango de 0 – 255 la función de transformación resulta ser:

$$q = 255 - p \quad (2)$$

Figura 3. 2 Operador inverso aplicado a figura 3.1.

3.1.3 Operador umbral.

Esta clase de transformación crea una imagen de salida binaria a partir de una imagen de grises, donde el nivel de transición está dado por el parámetro de entrada p1

Figura 3. 3 Operador umbral aplicado a la figura 3.1.

$$q = \begin{cases} 0\,, p\ < p1 \\ 255, p\ \geq p1 \end{cases} \quad (3)$$

3.1.4 Operador intervalo de umbral binario.

Esta clase de transformación crea una imagen de salida binaria a partir de una imagen de grises, donde todos los valores de gris cuyo nivel está en el intervalo definido por p1 y p2 son transformados a 255 y todos los valores fuera de ese intervalo a 0. La función de transformación es:

$$q = \begin{cases} 255\ para\ p \leq p1\ o\ p \geq p2 \\ 0\ para\ p1 < p < p2 \end{cases} \quad (4)$$

Figura 3. 4 Operador intervalo de umbral binario aplicado a la figura 3.1.

3.1.5 Operador intervalo de umbral binario invertido.

Esta clase de transformación crea una imagen de salida binaria a partir de una imagen de grises, donde todos los valores de gris cuyos niveles están en el intervalo definido por p1 y p2 son transformados a 0 y todos los valores fuera de ese intervalo son transformados a 255. La función de transformación es la siguiente:

$$q = \begin{cases} 0\ para\ p \leq p1\ o\ p \geq p2 \\ 255\ para\ p1 < p < p2 \end{cases} \quad (5)$$

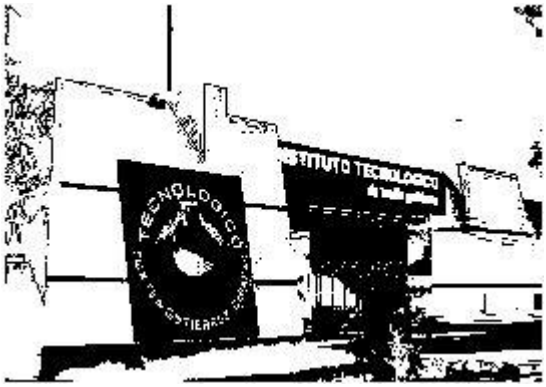


Figura 3. 5 Operador intervalo de umbral binario invertido aplicado a la figura 3.1.

3.1.6 Operador de umbral de la escala de grises.

Esta clase de transformación crea una imagen de salida con los valores de nivel de gris comprendidos en el intervalo definido por p1 y p2 y el resto a 255. La función de transformación es la siguiente:

$$q = \begin{cases} 255\ para\ p \leq p1\ o\ p \geq p2 \\ p\ para\ p1 < p < p2 \end{cases} \quad (6)$$

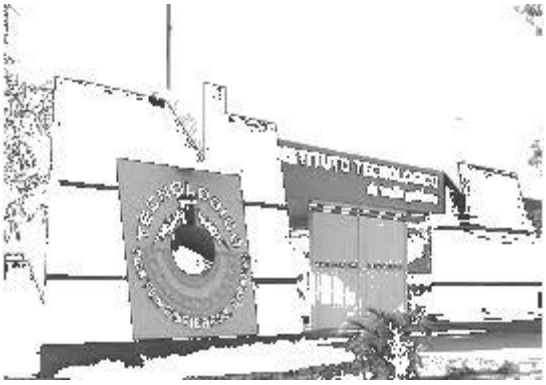


Figura 3. 6 Operador de umbral de la escala de grises aplicado a la figura 3.1.

3.1.7 Operador de umbral de la escala de grises invertido.

Esta clase de transformación crea una imagen de salida con los únicos valores de nivel de gris invertidos comprendidos en el intervalo definido por p1 y p2 y el resto a 255. La función de transformación es la siguiente:

$$q = \begin{cases} 255\ para\ p \leq p1\ o\ p \geq p2 \\ 255 - p\ para\ p1 < p < p2 \end{cases} \quad (7)$$

Figura 3. 7 Operador de umbral de la escala de grises invertido aplicado a la figura 3.1.

3.1.8 Operador de extensión.

Esta clase de operadores proporciona una imagen de salida con la escala de grises completa correspondiente al intervalo de entrada definido por p1 y p2 y suprime todos los valores fuera de este rango. La función de transformación es:

$$q = \begin{cases} 0 \; para \; p \leq p1 \; o \; p \geq p2 \\ (p - p1)\left(\frac{255}{p2 - p1}\right) para \; p1 < p < p2 \end{cases} \quad (8)$$

Figura 3. 8 Operador extensión aplicado a la figura 3.1.

3.1.9 Operador reducción del nivel de gris.

Esta clase de operadores proporciona una imagen de salida con un número de niveles de gris respecto de la imagen original de entrada, la imagen de entrada es reducida a n+1 niveles de gris con la siguiente transformación:

$$q = \begin{cases} 0 \; para \; p \leq p1 \\ q1 \; para \; p1 < p \leq p2 \\ qn \; para \; p_{n-1} < p < 255 \end{cases} \quad (9)$$

Figura 3. 9 Operador reducción del nivel de gris aplicado a la figura 3.1.

3.2 Operadores de vecindad.

Las operaciones de vecindad utilizan el mismo procedimiento excepto que el nuevo valor del píxel en la imagen de salida depende de una combinación de los valores de los píxeles en la vecindad del píxel de la imagen original que está siendo transformada. Básicamente consiste en transformar el valor de un píxel *p* en la posición (*x,y*) teniendo en cuenta los valores de los píxeles vecinos.

3.3 Modelos de optimización.

Un algoritmo genético (AG), es una técnica de programación computacional que imita a la evolución biológica como estrategia para la resolución de problemas. Dado un problema específico a resolver, la entrada del AG es un conjunto de soluciones potenciales a ese problema, codificadas de alguna manera específica, y que permite evaluar cuantitativamente a cada opción.

Esta evaluación de cada solución candidata es hecha de acuerdo con una función de aptitud o función objetivo. En un acervo de candidatas generadas aleatoriamente la mayoría no funcionarán en absoluto, y serán eliminadas, pero habrá otras que pueden ser prometedoras hacia la solución de problemas. Estas candidatas prometedoras se

conservarán realizándose múltiples copias de ellas. Estas copias no son perfectas, se introducen cambios pseudoaleatorios durante el proceso de copia, formando una nueva población de soluciones candidatas.

Esta nueva generación de soluciones candidatas son sometidas a una nueva ronda de evaluación de aptitud. Las candidatas que han empeorado o no han mejorado con los cambios son eliminadas. Pero, al realizar variaciones aleatorias introducidas en la población se puede haber mejorado el desempeño de alguna de las posibles candidatas, convirtiéndolas en mejores soluciones del problema, más completas o más eficientes.

De ésta forma, se seleccionan y copian estos individuos vencedores hacia la siguiente generación promoviendo en ellos cambios aleatorios, repitiendo el proceso hasta una cierta condición de término, figura 3.11**.**

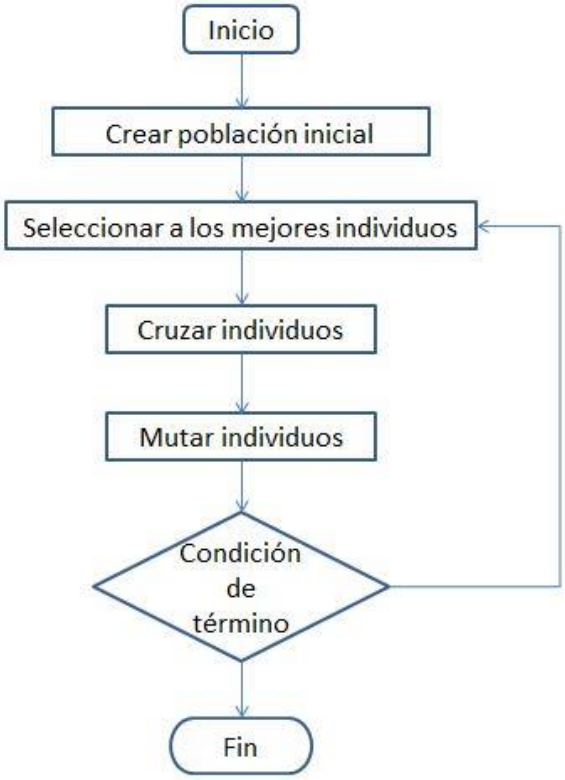


Figura 3. 10 Diagrama a bloques de un Algoritmo Genético.

Para escribir el algoritmo genético simple usé el lenguaje de programación Java y el entorno de desarrollo integrado de Eclipse. A continuación se explicarán brevemente cada uno de los componentes de dicha estructura.

3.3.1 Cromosoma y función de aptitud.

El programa nos pide los datos necesarios para crear la población inicial con la que el algoritmo trabajará. Primeramente, nos muestra un menú con algunas funciones, en la cual debemos escoger con cuál queramos trabajar:

```
Menú:
(1) f(x) = x^2
(2) f(x) = -0.1x^2 + 80x - 150
(3) f(x) = -(x-350)^2 + 350x
(4) f(x) = (x - 350)^2
(5) f(x) = (80 - (x - 25)^2) * exp(- (x - 25)^2 / 5) + 125
(6) f(x) = (x-1000)^2 sin(x)
Escoja una función:             1
```

Figura 3. 11 Menú de funciones en el programa.

Con éste objetivo escribimos el método menú()

```
320    private static void Menu() {
321        System.out.print("Menú:\n(1) f(x) = x^2\n(2) f(x) = -0.1x^2 + 80x - 150\n(3) f(x) = -(x-350)^2 + 350x\n"
322                + "(4) f(x) = (x - 350)^2\n(5) f(x) = (80 - (x - 25)^2) * exp(- (x - 25)^2 / 5) + 125\n"
323                + "(6) f(x) = (x-1000)^2 sin(x)\n"
324                + "Escoja una función:\t\t");
325    }
```

Figura 3. 12 Método menú()

La instrucción System.out.print() despliega en pantalla el texto que hayamos escrito entre los paréntesis.

Luego, nos solicita ingresar los datos del número de individuos (soluciones posibles al problema) y el número de genes de los individuos (bits de la cadena de soluciones):

```
Número de individuos:           10
Número de genes por individuo:  10
Número de iteraciones:          50
```

Figura 3. 13 Petición de datos para crear la población inicial.

Después nos despliega la población generada pseudoaleatoriamente, así como algunos datos que serán útiles durante el progreso del algoritmo:

```
[ 0 0 1 1 1 0 0 1 0 1 ]   [ 229.0 ]   [ 52441.0 ]   [ 0.015977028128356094 ]   [ 0.015977028128356094 ]
[ 1 1 1 0 0 1 1 1 0 0 ]   [ 924.0 ]   [ 853776.0 ]   [ 0.26011714435871464 ]   [ 0.27609417248707074 ]
[ 1 0 0 0 1 1 1 1 0 1 ]   [ 573.0 ]   [ 328329.0 ]   [ 0.1000309236733668 ]   [ 0.37612509616043754 ]
[ 0 1 1 0 0 0 0 1 0 1 ]   [ 389.0 ]   [ 151321.0 ]   [ 0.04610247465553618 ]   [ 0.42222757081597373 ]
[ 0 0 0 0 0 0 0 0 0 1 ]   [ 1.0 ]     [ 1.0 ]       [ 3.046667326777921E-7 ]   [ 0.4222278754827064 ]
[ 0 0 1 0 0 1 0 1 1 0 ]   [ 150.0 ]   [ 22500.0 ]   [ 0.006855001485250322 ]   [ 0.4290828769679567 ]
[ 1 0 1 1 1 1 0 1 0 0 ]   [ 756.0 ]   [ 571536.0 ]   [ 0.17412800572773457 ]   [ 0.6032108826956912 ]
[ 0 0 1 0 0 1 1 1 1 1 ]   [ 159.0 ]   [ 25281.0 ]   [ 0.007702279668827261 ]   [ 0.6109131623645185 ]
[ 1 1 0 0 1 0 1 0 1 1 ]   [ 811.0 ]   [ 657721.0 ]   [ 0.2003857080835701 ]   [ 0.8112988704480886 ]
[ 1 1 0 0 0 1 0 0 1 1 ]   [ 787.0 ]   [ 619369.0 ]   [ 0.1887011295519114 ]   [ 1.0 ]

Suma:                   3282275.0
Media:                  328227.5
Desviación Estándar:    304805.2208602241
Máximo:                 853776.0
Mínimo:                 1.0
Error:                  96387.87406289755
```

Figura 3. 14 Población inicial, decodificación de cromosomas y datos de dispersión.

En la primera columna podemos ver la población generada por el algoritmo; utilizando los datos ingresados por teclado del número de individuos y el número de genes construimos un arreglo bidimensional tipo entero.

```
45          int[][] poblacion = new int[individuos][variables * genes];
```

Posteriormente lo llenamos con valores binarios. Para esto, llamamos al método PoblacionInicial().

```
305    private static void PoblacionInicial(int[][] poblacion) {
306        Random aleatorio = new Random();
307
308        int filas = poblacion.length;
309        int columnas = poblacion[0].length;
310
311        for (int i = 0; i < filas; i++) {
312            for (int j = 0; j < columnas; j++) {
313                int binario = aleatorio.nextInt(2);
314
315                poblacion[i][j] = binario;
316            }
317        }
318    }
```

Figura 3. 15 Generación de la población inicial.

Al método le pasamos el parámetro población previamente creado y mediante ciclos for rellenamos cada celda del arreglo. Para generar los números importamos la clase *java.util.Random*, creamos un objeto llamado aleatorio y usamos el método nextInt(), a éste último le damos el parámetro de 2 para que genere números entre 0 y 1.

La segunda columna nos dice el valor decodificado de la cadena binaria de cada individuo. El algoritmo utilizado para convertir del sistema binario al sistema decimal fue el siguiente:

```
232            for (int j = 0; j < variables; j++) {
233                int pot = genes - 1;
234                double decodificador = 0;
235
236                for (int k = 0; k < columnas; k++) {
237                    decodificador += poblacion[i][k] * Math.pow(2, pot);
238                    if (pot == 0) {
239                        calculos[i][j] = decodificador;
240                        pot = genes - 1;
241                        decodificador = 0;
242                    }
243                    pot--;
244                }
```

Figura 3. 16 Algoritmo para la decodificación de cromosomas.

Con ayuda de las variables llamadas *pot* y *decodificador* conseguimos la conversión de cada cadena binaria a valores decimales; en el ciclo for nos desplazamos de izquierda

a derecha multiplicando el contenido de cada celda por dos elevado al valor contenido en la variable *pot* y lo sumamos a la variable *decodificador*. La variable *pot* almacena el número de la posición de cada bit desde el mayor menos uno hasta cero. Al valor obtenido en la conversión se le llama aptitud.

En la tercera columna se encuentra el valor de la función objetivo, ésta hace referencia a la función que vamos a optimizar. Para éste caso es:

$$f(x) = x^2 \quad (10)$$

Con los valores de ésta columna se obtienen los siguientes conceptos encontrados en la parte inferior de la figura 3.15:

$$Suma = \sum_{n=1}^{N} (x_n) \quad (11)$$

$$Media = \frac{suma}{N} \quad (12)$$

$$Desviación\ Estándar = \sqrt{\frac{\sum (x_n - Media)^2}{N}} \quad (13)$$

$$Error = \frac{Desviación\ Estándar}{\sqrt{N}} \quad (14)$$

donde: x_n, el valor de aptitud del individuo.
N, el total de individuos en la población.

Después, la cuarta columna nos muestra la probabilidad de selección del individuo, esto lo obtenemos con:

$$Probabilidad\ de\ selección = \frac{f(x)}{\sum f(x)} \quad (15)$$

La quinta columna es la probabilidad de selección acumulada. Los datos de la parte inferior se utilizan para generar gráficas y analizar el funcionamiento del algoritmo.

3.3.2 Selección de los mejores individuos.

El operador de selección escoge los mejores individuos de la población mediante el método de la ruleta para su posterior cruce.

```
177             for (int padre = 0; padre < 2; padre++) {
178                 int eleccion = 0;
179                 double ruleta = aleatorio.nextDouble();
180                 for (int seleccion = 0; ruleta > calculos[seleccion][variables + 2]; seleccion++) {
181                      eleccion = seleccion + 1;
182                 }
183                 for (int gen = 0; gen < columnas; gen++) {
184                     padres[padre][gen] = poblacion[eleccion][gen];
185                 }
186             }
```

Figura 3. 17 Operador de selección con método ruleta.

El primer ciclo *for* está limitado a escoger dos individuos de la población. El segundo *for* es quién simula una ruleta, de nuevo utilizando la clase *random*, pero esta vez el método *nextDoble()*, generamos un número aleatorio tipo *double* que está en el rango de 0 a 1, lo almacena en la variable *ruleta* y lo compara con cada uno de los datos de la columna de probabilidad de selección acumulada. Mientras el valor generado sea menor al valor de la probabilidad de selección acumulada se continúa en el ciclo y se aumenta la variable elección. Cuando la condición del ciclo se vuelve verdadera y termina, el valor que queda almacenado en *elección* es el índice del individuo seleccionado.

El siguiente ciclo for copia el contenido de los dos individuos seleccionados para el siguiente operador.

3.3.3 Cruce.

El operador cruce es el encargado de combinar la información de los dos individuos seleccionados en el operador selección. Utiliza la técnica de cruce de un punto.

```
188             double probabilidadcruce = 0.8;
189             double probabilidad = aleatorio.nextDouble();
190             if (probabilidad < probabilidadcruce) {
191                 int pcruce1 = aleatorio.nextInt(columnas - 1);
192                 for (int gen = 0; gen < columnas; gen++) {
193                     if (gen < pcruce1) {
194                         poblacion[n][gen] = padres[0][gen];
195                     } else {
196                         poblacion[n][gen] = padres[1][gen];
197                     }
198                 }
```

Figura 3. 18 Operador cruce.

Ajustamos la probabilidad de que se realice el cruce en 0.8, luego generamos un valor aleatorio y lo almacenamos en la variable *probabilidad*, si la condición se cumple, genera un numero aleatorio entero que será el bit que indique el punto donde serán divididos los individuos y se combinará la información para generar un nuevo individuo.

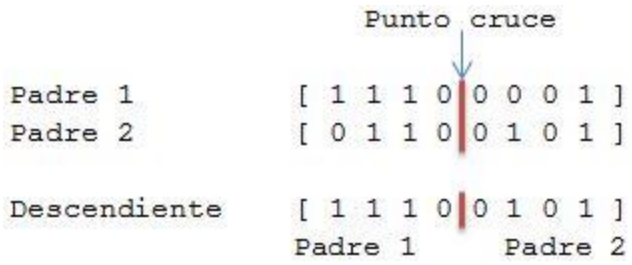


Figura 3. 19 Recombinación de información entre individuos con el operador cruce.

3.3.4 Mutación.

El operador mutación está configurado con probabilidad de 0.005 y se realiza bit a bit.

```
155         double pmutacion = 0.005;
156
157         for (int i = 0; i < filas; i++) {
158             for (int j = 0; j < columnas; j++) {
159                 double probabilidad = aleatorio.nextDouble();
160                 if (probabilidad < pmutacion) {
161                     poblacion[i][j] = Math.abs(poblacion[i][j] - 1);
162                 }
163             }
164         }
```

Figura 3. 20 Operador mutación.

Esto significa que por cada bit en el contenido del individuo se genera un numero aleatorio tipo double con el método *nextDouble()* y es comparado con la probabilidad de mutación, si es menor, se cambia el bit en donde se encuentren los ciclos anidados for. Si la condición no es cierta, se pasa a la siguiente celda sin modificar el contenido de la celda anterior.

Finalmente, el programa muestra la población final, en este caso la generación número cincuenta; al ser la función objetivo una parábola, los valores deben ser cada vez más grandes, de hecho, al haber ingresado el 10 como la cantidad de genes, deben tender a 1 048 576, que es el resultado de elevar 1024 a la potencia dos:

```
[ 1 1 1 0 0 0 1 1 1 0 ]    [ 910.0 ]    [ 828100.0 ]    [ 0.09808403886880594 ]    [ 0.09808403886880594 ]
[ 1 1 1 0 0 1 1 1 1 0 ]    [ 926.0 ]    [ 857476.0 ]    [ 0.10156346976581118 ]    [ 0.1996475086346171 ]
[ 1 1 1 0 0 1 1 1 1 0 ]    [ 926.0 ]    [ 857476.0 ]    [ 0.10156346976581118 ]    [ 0.3012109784004283 ]
[ 1 1 1 0 0 0 1 1 1 0 ]    [ 910.0 ]    [ 828100.0 ]    [ 0.09808403886880594 ]    [ 0.3992950172692343 ]
[ 1 1 1 0 0 0 1 1 1 0 ]    [ 910.0 ]    [ 828100.0 ]    [ 0.09808403886880594 ]    [ 0.49737905613804023 ]
[ 1 1 1 0 0 1 1 1 1 0 ]    [ 926.0 ]    [ 857476.0 ]    [ 0.10156346976581118 ]    [ 0.5989425259038514 ]
[ 1 1 1 0 0 0 0 1 1 0 ]    [ 902.0 ]    [ 813604.0 ]    [ 0.09636706479871511 ]    [ 0.6953095907025666 ]
[ 1 1 1 0 0 1 1 1 1 0 ]    [ 926.0 ]    [ 857476.0 ]    [ 0.10156346976581118 ]    [ 0.7968730604683778 ]
[ 1 1 1 0 0 1 1 1 1 0 ]    [ 926.0 ]    [ 857476.0 ]    [ 0.10156346976581118 ]    [ 0.898436530234189 ]
[ 1 1 1 0 0 1 1 1 1 0 ]    [ 926.0 ]    [ 857476.0 ]    [ 0.10156346976581118 ]    [ 1.0000000000000002 ]
        Suma:                   8442760.0
        Media:                  844276.0
        Desviación Estándar:    16646.92317516964
        Máximo:                 857476.0
        Mínimo:                 813604.0
        Error:                  5264.219326737821
```

Figura 3. 21 Población final.

Existe una parte del código que se dedica a generar las ventanas con información que puede servir para entender el comportamiento del algoritmo.

En la siguiente ventana, se muestra información de la generación cero, es decir, la población que el algoritmo genera aleatoriamente, la línea roja nos muestra la gráfica de la función que estamos trabajando y con los puntos azules indicamos cada uno de los individuos de dicha población.

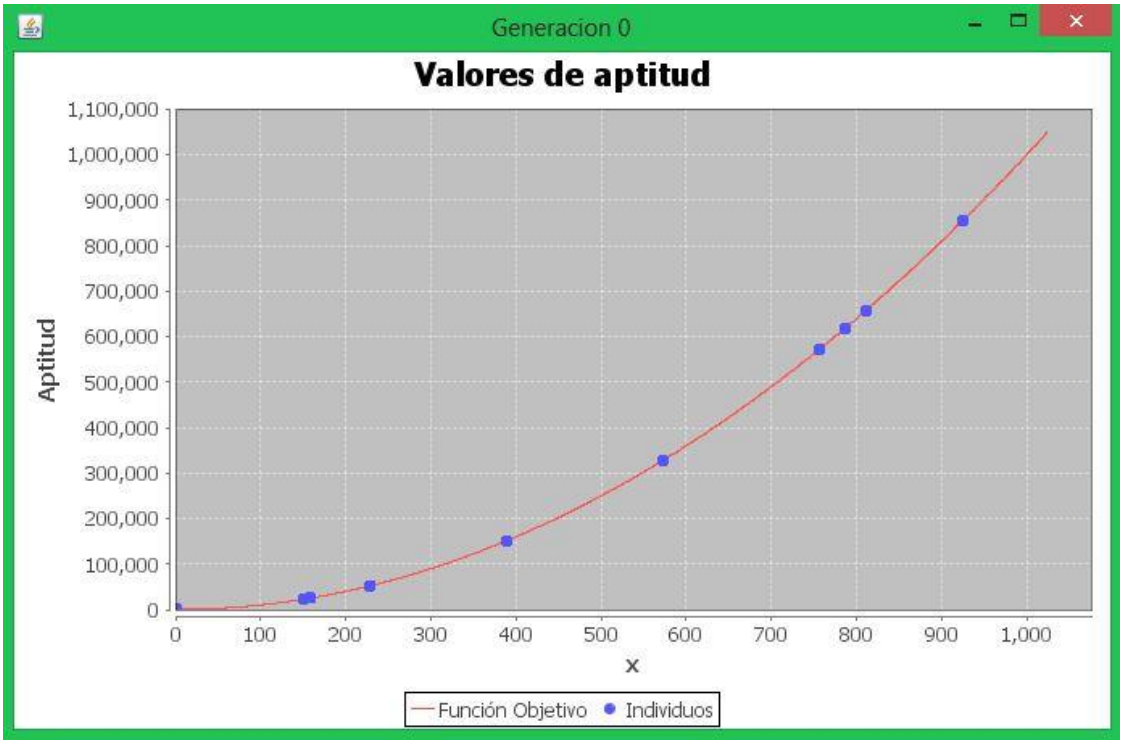


Figura 3. 22 Gráfica de la función objetivo e individuos de la población inicial.

En este caso, podemos notar que existen los 10 individuos que elegimos para este problema, quiere decir, que cada uno de los individuos tiene un valor diferente en su decodificación; pero existe el caso en que, se observen menos puntos debido a que uno quede dibujado sobre otro porque tienen el mismo valor de decodificación.

Para la generación número 12, el valor de los individuos es el mismo y se trasponen unos a otros. Los individuos con menor valor de adaptación se han eliminado, por el contrario, los que sobrevivieron se utilizaron para crear nuevos individuos con valores mayores.

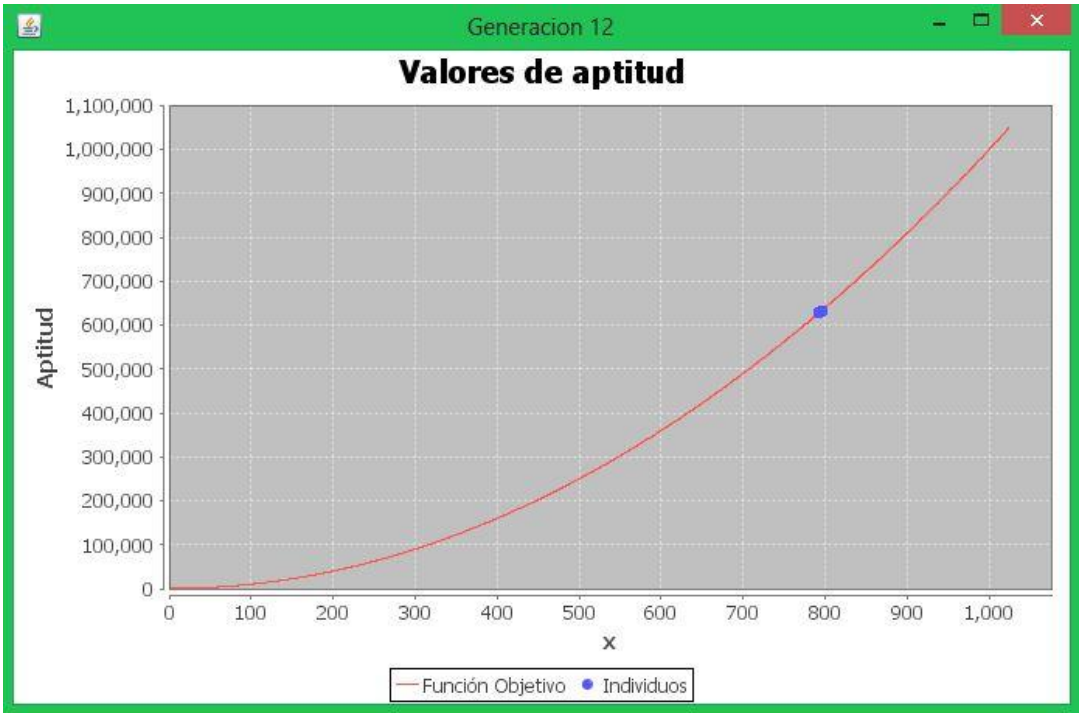


Figura 3. 23 Grafica de la generación 12.

En la generación 24, observamos la consecuencia del operador de cruce y mutación. Si estos operadores no existieran, en una población de individuos con el mismo valor de adaptación, no habría diversidad en el material genético y no se producirían cambios.

Es aquí donde los algoritmos genéticos toman mayor importancia como alternativas para la resolución de problemas. El operador de cruce combina el material codificado de cada individuo manteniendo así la diversidad en la población. El operador de mutación permite eso cambios esporádicos que aumentan o disminuyen los valores de adaptación, que posteriormente sobrevivirán para la siguiente generación o serán eliminados respectivamente.

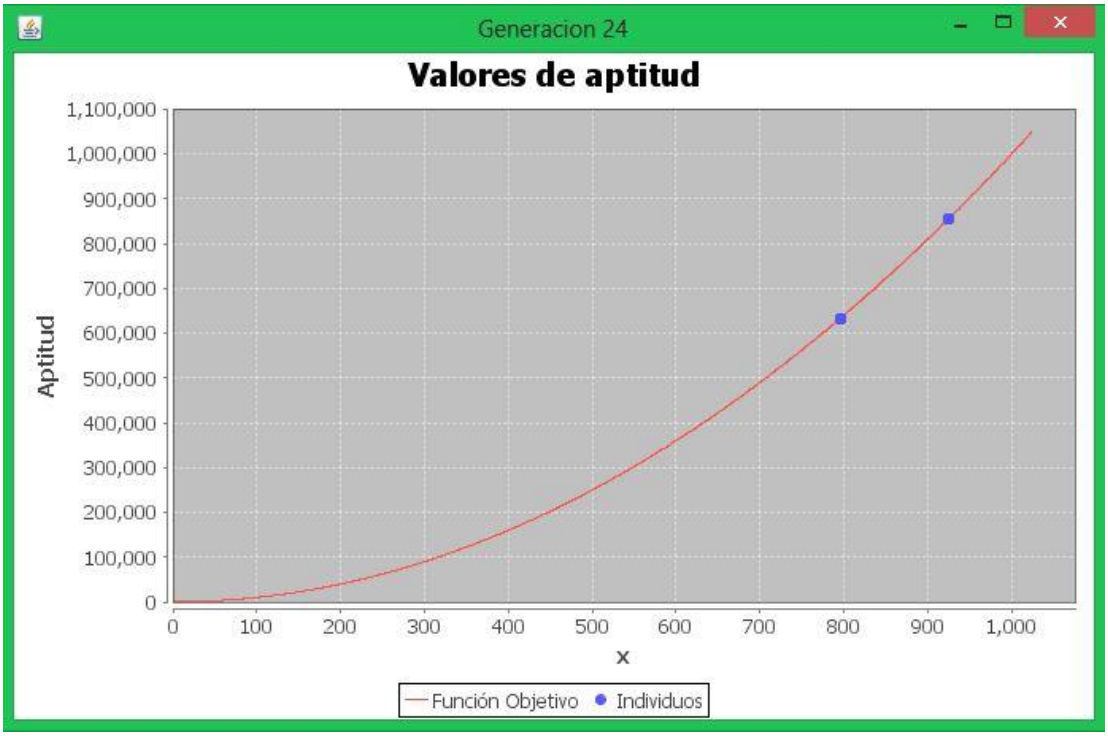


Figura 3. 24 Grafica de la generación 24.

Durante la generación número 36 y 48, los individuos vuelven a trasponerse entre ellos y cada vez maximizan más los valores de la función objetivo. En el tema de resultados se anexarán los datos que se obtienen de las demás funciones del menú mostrado al inicio de este tema.

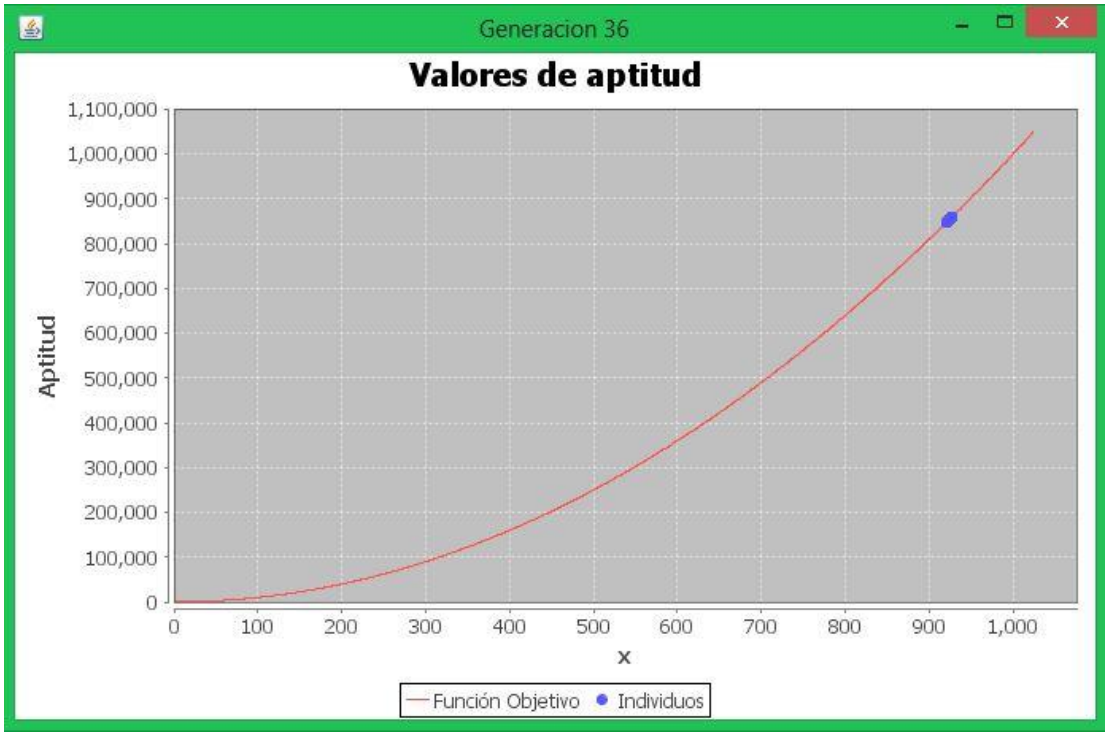


Figura 3. 25 Gráfica de la generación 36.

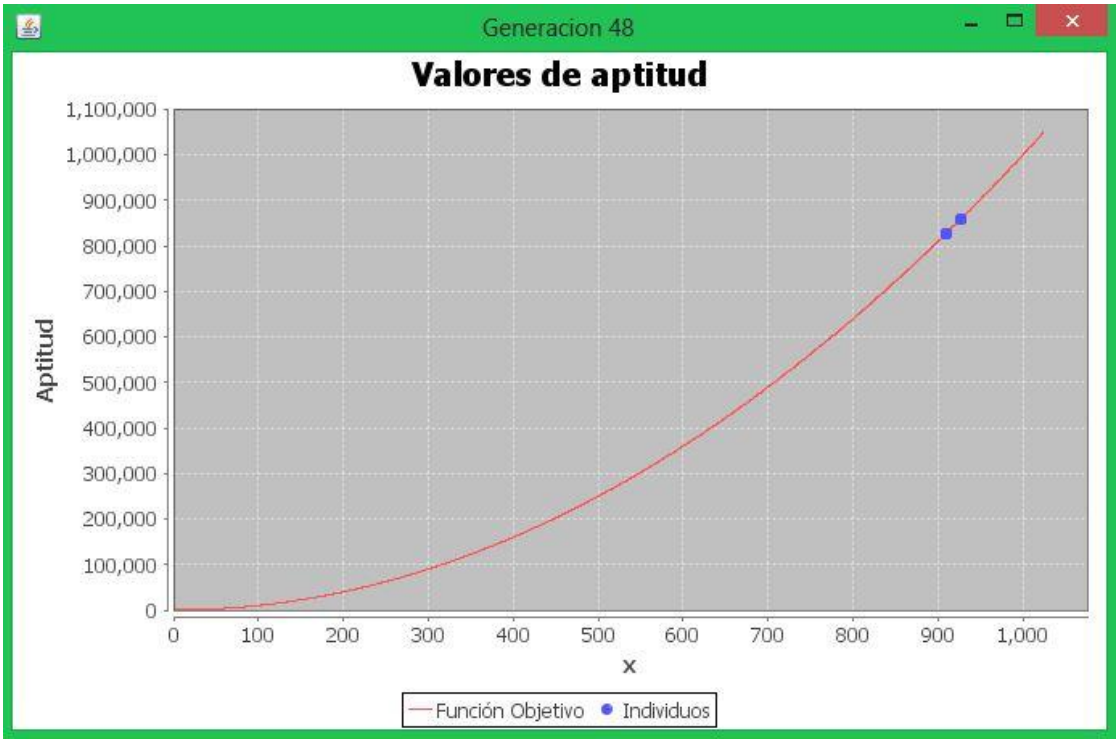


Figura 3. 26 Gráfica de la generación 48.

3.4 Resolución de problemas de optimización con Algoritmos Genéticos.

Como se menciona anteriormente, los algoritmos genéticos son aplicados en la solución de problemas donde existen muchas variables. Esta vez, aumentaremos la complejidad a dos variables, con la intención de resolver el problema de diseño de una lata.

Para la situación en la que la lata es cilíndrica, podemos definir dos parámetros: el diámetro “d” y la altura “h”. Además, necesitamos saber el volumen, que será de 300 ml.

El objetivo del algoritmo genético será el de minimizar el costo para el material del que será fabricada la lata. Si quisiéramos que el material fuera aluminio; en nuestro país se pueden conseguir placas de 122 cm x 244 cm, con precios que van desde los 1600 pesos hasta los 2000 pesos. Para nuestro algoritmo, usaremos un costo de 0.065 pesos por cm^2.

Con el objetivo y la restricción ya definidos escribimos el correspondiente algoritmo genético.

Minimizar (objetivo)

$$f(d,h) = c\left[\frac{\pi d^2}{2} + \pi dh\right] \qquad (16)$$

Sujeto a (restricción)

$$g(d,h) \leftrightarrow \frac{\pi d^2 h}{4} \geq 300 \qquad (17)$$

Para codificar las variables d y h usamos 5 bits por cada una, entonces tenemos cadenas binarias de 10 bits.

Utilizando la misma metodología, primero generamos aleatoriamente una población inicial y calculamos la respectiva información para cada individuo.

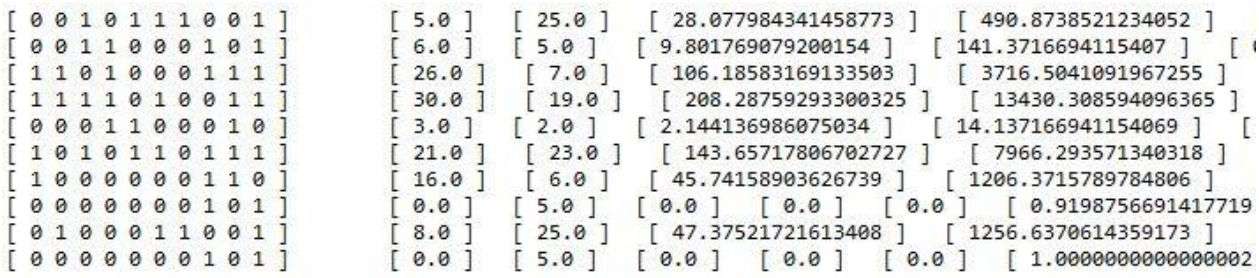


Figura 3. 27 Población inicial, función objetivo y restricción.

En la primera columna podemos observar la población generada aleatoriamente. La segunda nos muestra el valor de la variable d y la tercera columna imprime el valor de h. La siguiente columna nos informa del costo de producir la lata con los valores de las dos columnas anteriores. La última columna nos indica la capacidad que puede contener esa lata.

Por ejemplo, para el elemento de la segunda fila, el valor de la variable d son los primeros cinco números de izquierda a derecha y es igual a 6. El valor de la variable h son los cinco siguientes números y es igual a 5.

Con estos dos valores obtenemos que el costo de producción es de 9.8017 pesos y tiene una capacidad de 141.37 ml.

Cabe mencionar que para este elemento, se aplica una penalización debido a que el volumen es menor al que deseamos. La penalización consiste en sumar una cantidad lo suficientemente grande para causar que todas las soluciones inútiles tengan valores peores que los que están en alguna solución útil. Los demás elementos no son alterados pues cumplen con la restricción.

El siguiente paso es aplicar los operadores de selección, cruce y mutación un número determinado de iteraciones; esta ocasión se utilizó para la selección de individuos el método de torneo y fueron 100 repeticiones.

Al finalizar el trabajo del algoritmo, obtenemos los siguientes datos:

[0 0 1 0 0 1 1 0 0 0]	[4.0]	[24.0]	[21.237166338267006]	[301.59289474462014]
[0 0 1 0 0 1 1 0 0 0]	[4.0]	[24.0]	[21.237166338267006]	[301.59289474462014]
[0 0 1 0 0 1 1 0 0 0]	[4.0]	[24.0]	[21.237166338267006]	[301.59289474462014]
[0 0 1 0 0 1 1 0 0 0]	[4.0]	[24.0]	[21.237166338267006]	[301.59289474462014]
[0 0 1 0 0 1 1 0 0 0]	[4.0]	[24.0]	[21.237166338267006]	[301.59289474462014]
[0 0 1 0 0 1 1 0 0 0]	[4.0]	[24.0]	[21.237166338267006]	[301.59289474462014]
[0 0 1 0 0 1 1 0 0 0]	[4.0]	[24.0]	[21.237166338267006]	[301.59289474462014]
[0 0 1 0 0 1 1 0 0 0]	[4.0]	[24.0]	[21.237166338267006]	[301.59289474462014]
[0 0 1 0 0 1 1 0 0 0]	[4.0]	[24.0]	[21.237166338267006]	[301.59289474462014]
[0 0 1 0 0 1 1 0 0 0]	[4.0]	[24.0]	[21.237166338267006]	[301.59289474462014]

Figura 3. 28 Población final, parámetros de la lata optimizados.

En esta ocasión notamos que para un aproximado de 300 ml, se necesita un prisma cilíndrico con medidas de 4cm de diámetro y 24cm de alto con un costo de 21.23 pesos.

3.5 Resultados, gráficas y programas.

3.5.1 Resultados de los modelos de optimización.

Resultados y graficas de la función $f(x) = -0.1x^2 + 80x - 150$

Individuos: 30 Genes: 10 Iteraciones: 60

Población Inicial:

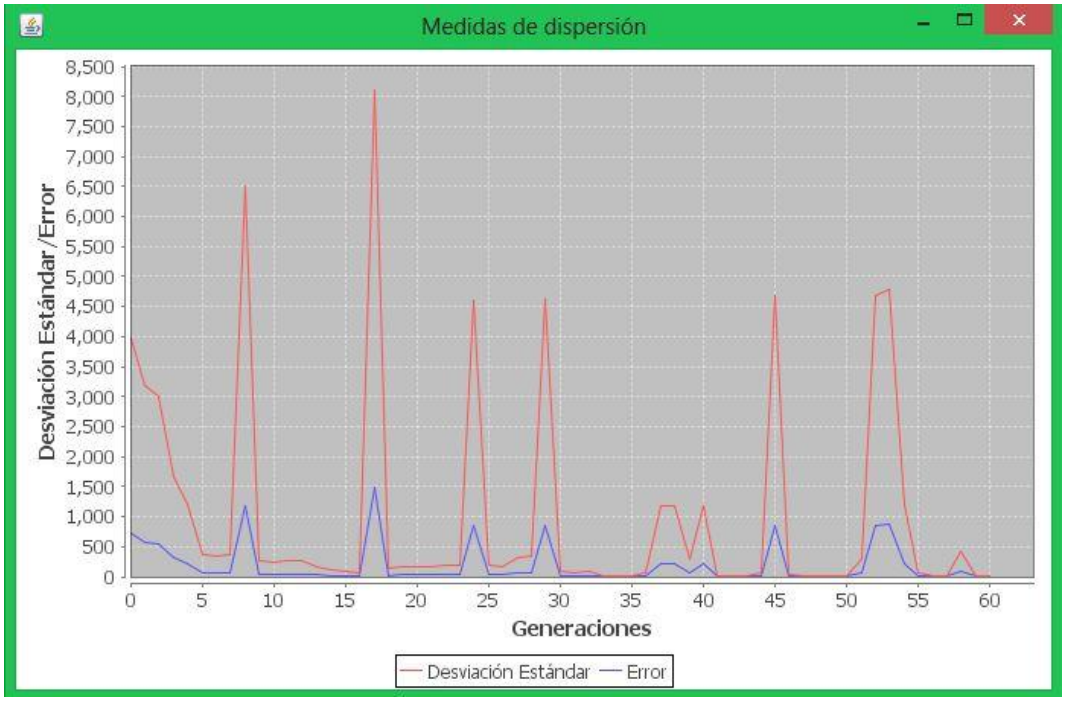
[0 0 0 0 1 1 0 0 1 0]	[50.0]	[3600.0]	[0.016385839175719466]	[0.016385839175719466]
[0 1 0 0 1 0 1 1 1 1]	[303.0]	[14909.1]	[0.0678605874596442]	[0.08424642663536366]
[0 0 1 1 1 0 0 0 1 0]	[226.0]	[12822.4]	[0.058362717846318135]	[0.1426091444816818]
[1 1 1 0 0 0 1 1 0 0]	[908.0]	[-9956.400000000009]	[-0.045317769213648176]	[0.09729137526803362]
[0 1 1 0 0 1 0 0 1 1]	[403.0]	[15849.099999999999]	[0.07213911213330426]	[0.16943048740133787]
[0 0 1 0 1 0 0 0 0 0]	[160.0]	[10090.0]	[0.045925865911947056]	[0.21535635331328493]
[1 0 1 1 0 1 1 1 1 0]	[734.0]	[4694.399999999994]	[0.02136713428513816]	[0.2367234875984231]
[0 1 1 1 1 1 0 0 1 0]	[498.0]	[14889.599999999999]	[0.0677718308307757]	[0.3044953184291988]
[1 1 0 1 0 0 1 1 1 1]	[847.0]	[-4130.900000000009]	[-0.018802295291938803]	[0.28569302313725997]
[1 0 0 1 0 1 1 0 0 0]	[600.0]	[11850.0]	[0.053936720620076574]	[0.33962974375733657]
[0 0 1 1 1 1 0 0 1 0]	[242.0]	[13353.599999999999]	[0.0607805394491354]	[0.400410283206472]
[0 0 1 0 0 0 1 0 1 0]	[138.0]	[8985.6]	[0.04089905458259579]	[0.4413093377890678]
[1 1 1 0 0 1 1 1 0 0]	[924.0]	[-11607.600000000006]	[-0.05283340744891149]	[0.38847593034015626]
[1 0 1 1 1 1 1 0 1 0]	[762.0]	[2745.5999999999985]	[0.01249693334468204]	[0.4009728636848383]
[0 0 0 0 0 0 0 0 1 0]	[2.0]	[9.599999999999994]	[4.3695571135251886E-5]	[0.40101655925597357]
[0 0 0 1 0 1 1 0 0 1]	[89.0]	[6177.9]	[0.02811946551213258]	[0.42913602476810614]
[0 1 1 1 1 1 1 1 0 0]	[508.0]	[14683.599999999999]	[0.06683419670016509]	[0.49597022146827124]
[1 0 0 1 0 0 0 0 1 0]	[578.0]	[12681.599999999999]	[0.05772184946966777]	[0.553692070937939]
[1 1 0 1 0 0 0 1 1 0]	[838.0]	[-3334.4000000000087]	[-0.01517692837431087]	[0.5385151425636282]
[1 0 0 1 1 1 1 1 0 0]	[636.0]	[10280.399999999994]	[0.04679249473946286]	[0.5853076373030911]
[0 1 0 1 1 0 1 0 1 1]	[363.0]	[15713.099999999999]	[0.07152009154222154]	[0.6568277288453126]
[1 1 0 0 1 1 1 1 1 0]	[830.0]	[-2640.0]	[-0.012016282062194275]	[0.6448114467831183]
[0 1 1 1 0 0 0 1 1 1]	[455.0]	[15547.5]	[0.07076634294013845]	[0.7155777897232567]
[1 1 0 1 1 0 0 0 1 0]	[866.0]	[-5865.600000000006]	[-0.026697993963638943]	[0.6888797957596178]
[0 0 0 0 0 1 0 0 1 0]	[18.0]	[1257.6]	[0.005724119818717999]	[0.6946039155783358]
[1 0 1 0 0 1 0 1 1 0]	[662.0]	[8985.599999999999]	[0.04089905458259578]	[0.7355029701609316]
[0 0 1 1 0 1 0 0 1 1]	[211.0]	[12277.9]	[0.05588435967099056]	[0.7913873298319222]
[0 1 1 1 1 0 1 0 1 0]	[490.0]	[15040.0]	[0.06845639477856133]	[0.8598437246104835]
[0 1 1 0 0 1 0 1 1 1]	[407.0]	[15845.099999999999]	[0.07212090564533125]	[0.9319646302558148]
[0 1 1 1 1 0 1 1 1 1]	[495.0]	[14947.5]	[0.0680353697441852]	[1.0]

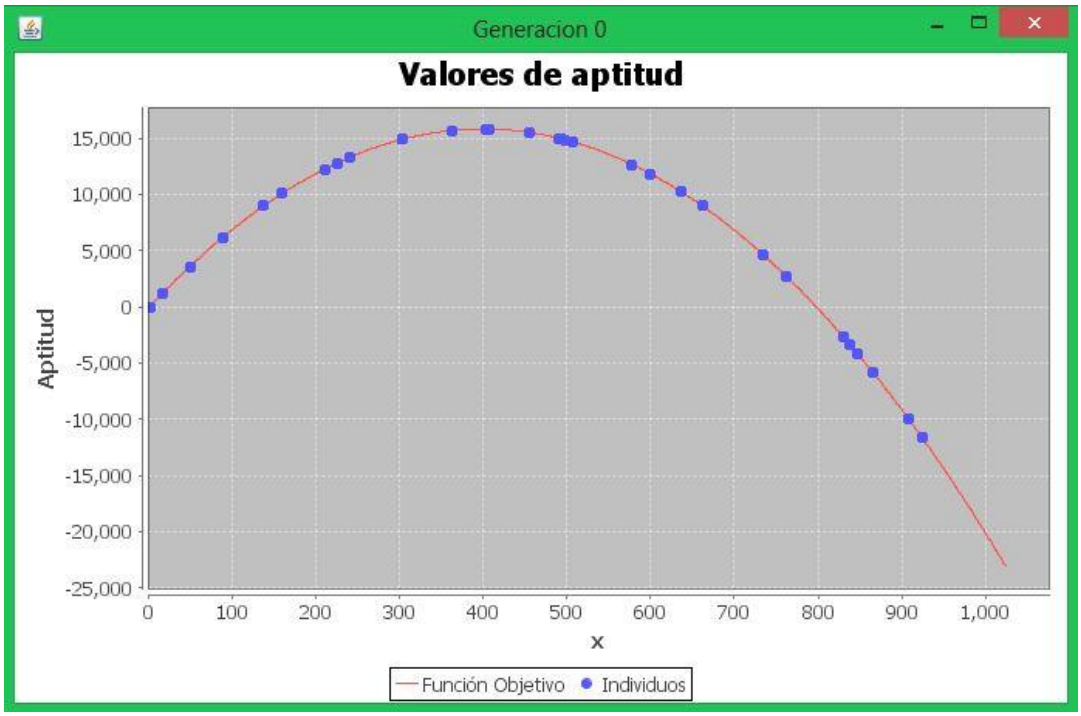
Suma:	219701.89999999997
Media:	7323.396666666666
Desviación Estándar:	8247.325458615189
Máximo:	15849.099999999999
Mínimo:	-11607.600000000006
Error:	1505.7487309233927

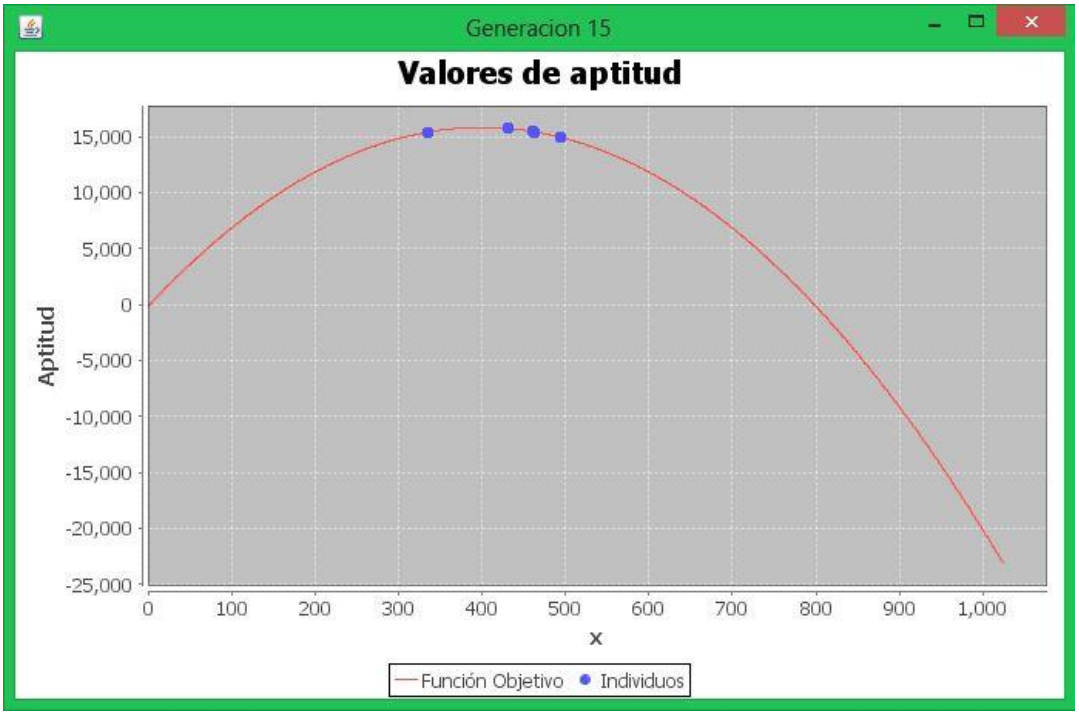
Población Final:

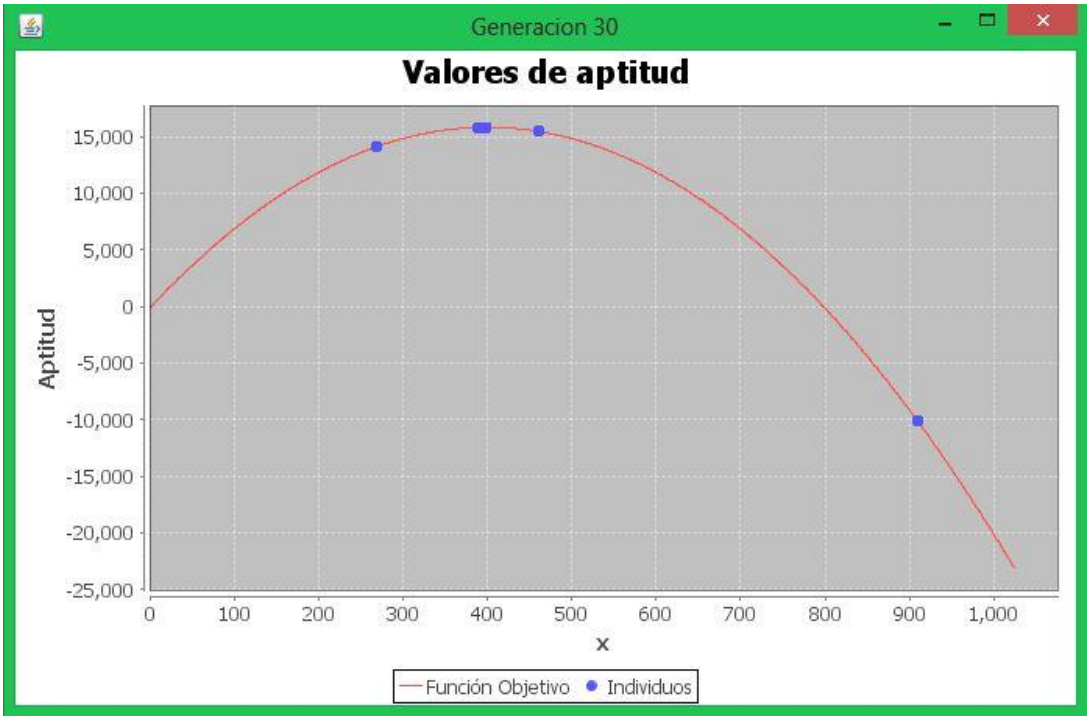
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.03333369085782534]
[0 1 1 0 0 0 1 1 1 0]	[398.0]	[15849.599999999999]	[0.033333059932251205]	[0.06666675079007654]
[0 1 1 0 0 0 1 1 1 0]	[398.0]	[15849.599999999999]	[0.033333059932251205]	[0.09999981072232775]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.13333350158015309]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.16666719243797842]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.20000088329580376]
[0 1 1 0 0 0 1 1 1 0]	[398.0]	[15849.599999999999]	[0.033333059932251205]	[0.23333394322805495]
[0 1 1 0 0 0 1 1 1 0]	[398.0]	[15849.599999999999]	[0.033333059932251205]	[0.26666700316030617]
[0 1 1 0 0 0 1 1 1 0]	[398.0]	[15849.599999999999]	[0.033333059932251205]	[0.3000000630925574]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.3333337539503827]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.36666744480820807]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.4000011356660334]
[0 1 1 0 0 0 1 1 0 0]	[396.0]	[15848.4]	[0.0333305362299547]	[0.4333316718959881]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.46666536275381343]
[0 1 1 0 0 0 1 1 1 0]	[398.0]	[15849.599999999999]	[0.033333059932251205]	[0.49999842268606465]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.53333211354389]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.5666658044017153]
[0 1 1 0 0 0 1 1 1 0]	[398.0]	[15849.599999999999]	[0.033333059932251205]	[0.5999988643339665]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.6333325551917919]
[0 1 1 0 0 0 1 1 1 0]	[398.0]	[15849.599999999999]	[0.033333059932251205]	[0.6666656151240431]
[0 1 1 0 0 0 1 1 1 0]	[398.0]	[15849.599999999999]	[0.033333059932251205]	[0.6999986750562943]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.7333323659141197]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.766666056771945]
[0 1 1 0 0 0 1 1 1 0]	[398.0]	[15849.599999999999]	[0.033333059932251205]	[0.7999991167041962]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.8333328075620215]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.8666664984198469]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[0.9000001892776722]
[0 1 1 0 0 0 1 1 1 0]	[398.0]	[15849.599999999999]	[0.033333059932251205]	[0.9333332492099234]
[0 1 1 0 0 0 1 1 1 0]	[398.0]	[15849.599999999999]	[0.033333059932251205]	[0.9666663091421747]
[0 1 1 0 0 0 1 1 1 1]	[399.0]	[15849.9]	[0.03333369085782534]	[1.0]

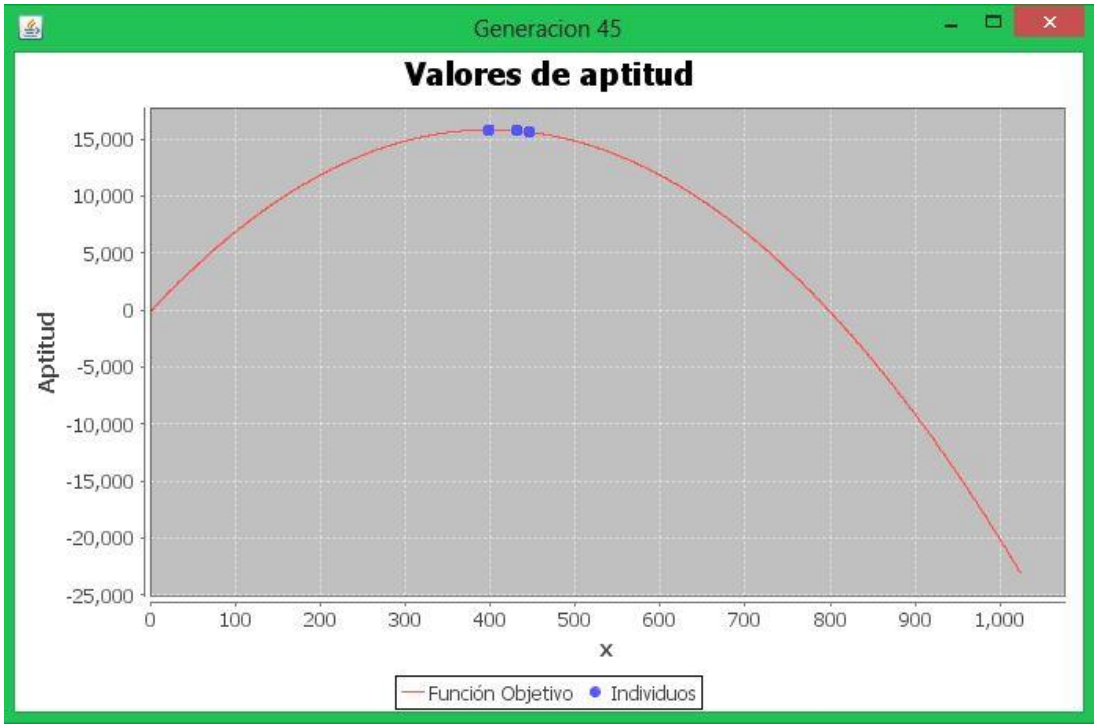
Suma:	475491.9
Media:	15849.730000000001
Desviación Estándar:	0.28653097563808616
Máximo:	15849.9
Mínimo:	15848.4
Error:	0.052313159593647664

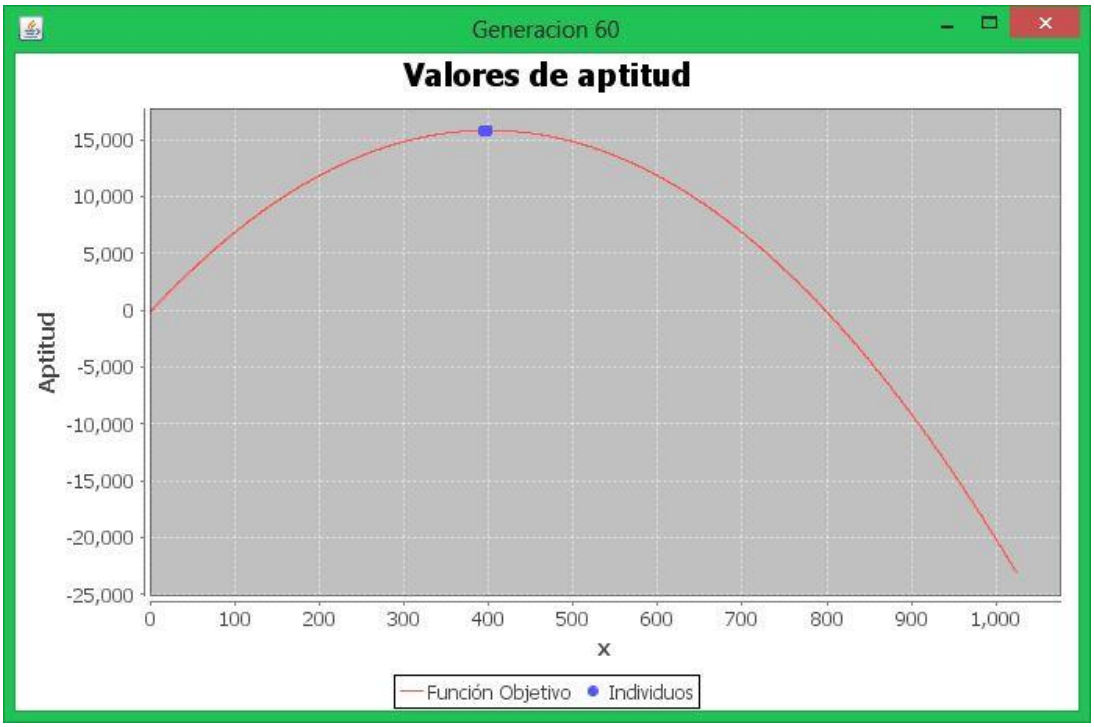

Medidas de dispersión
Desviación Estándar /Error
8,500
8,000
7,500
7,000
6,500
6,000
5,500
5,000
4,500
4,000
3,500
3,000
2,500
2,000
1,500
1,000
500
0
0
5
10
15
20
25
30
35
40
45
50
55
60
Generaciones
Desviación Estándar
Error


Generacion 0
Valores de aptitud
Aptitud
15,000
10,000
5,000
0
-5,000
-10,000
-15,000
-20,000
-25,000
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos


Generacion 15
Valores de aptitud
15,000
10,000
5,000
0
-5,000
-10,000
-15,000
-20,000
-25,000
Aptitud
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos


Generacion 30
Valores de aptitud
15,000
10,000
5,000
0
-5,000
-10,000
-15,000
-20,000
-25,000
Aptitud
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos


Generacion 45
Valores de aptitud
Aptitud
15,000
10,000
5,000
0
-5,000
-10,000
-15,000
-20,000
-25,000
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos


Generacion 60
Valores de aptitud
Aptitud
15,000
10,000
5,000
0
-5,000
-10,000
-15,000
-20,000
-25,000
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos

Resultados y graficas de la función $f(x) = -(x-350)^2 + 350x$

Individuos: 20 Genes: 10 Iteraciones: 100

Población Inicial:

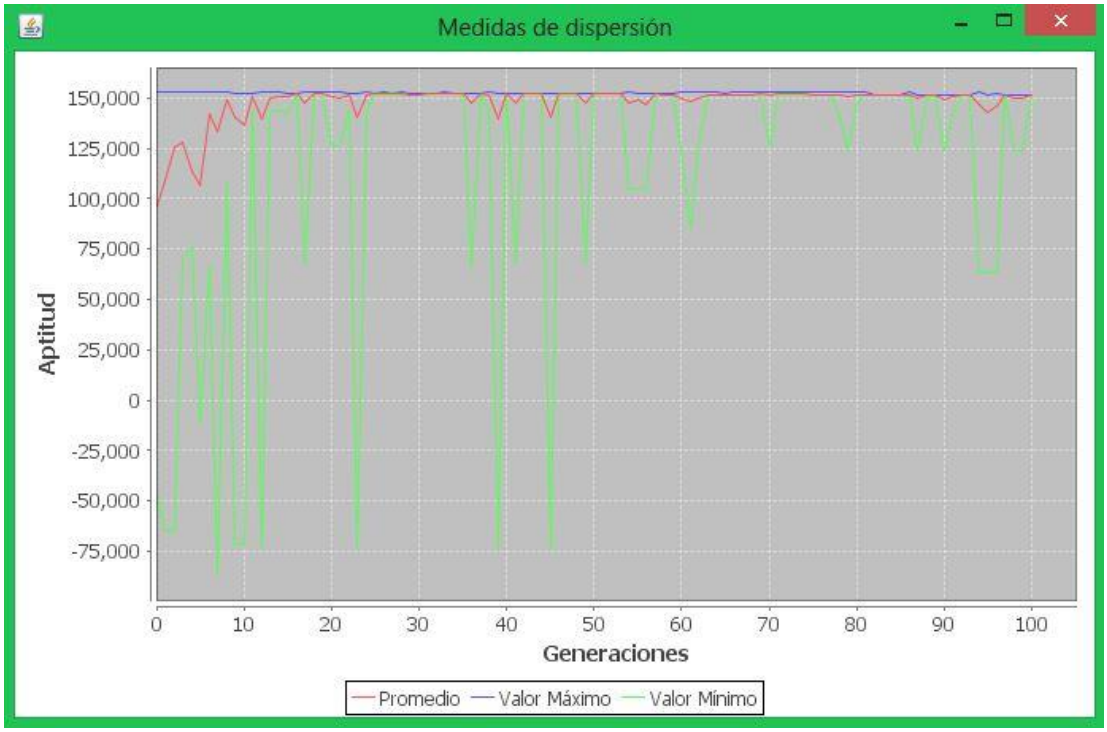
[0 1 1 0 1 1 1 0 0 0]	[440.0]	[145900.0]	[0.09877215180447041]	[0.09877215180447041]
[0 1 0 1 1 0 0 0 0 0]	[352.0]	[123196.0]	[0.08340187809255337]	[0.18217402989702378]
[1 1 0 1 1 0 0 0 0 1]	[865.0]	[37525.0]	[0.02540387249117719]	[0.20757790238820095]
[1 1 1 1 0 0 0 1 0 0]	[964.0]	[-39596.0]	[-0.026805908998285197]	[0.18077199338991576]
[1 1 0 1 0 0 0 0 1 1]	[835.0]	[57025.0]	[0.038605085377998116]	[0.21937707876791387]
[1 1 1 0 0 1 0 0 0 0]	[912.0]	[3356.0]	[0.0022719625870856936]	[0.22164904135499958]
[0 0 1 1 0 1 1 0 0 0]	[216.0]	[57644.0]	[0.039024139263995145]	[0.2606731806189947]
[1 0 1 0 1 0 0 1 0 1]	[677.0]	[130021.0]	[0.0880223026029407]	[0.3486954832219354]
[0 0 0 1 1 0 1 1 1 0]	[110.0]	[-19100.0]	[-0.012930418776322034]	[0.3357650644456134]
[1 0 0 1 0 1 0 0 0 1]	[593.0]	[148501.0]	[0.10053299050798944]	[0.43629805495360285]
[1 1 0 0 0 1 0 0 0 1]	[785.0]	[85525.0]	[0.057899165751044084]	[0.49419722070464694]
[0 1 0 0 0 1 0 0 1 0]	[274.0]	[90124.0]	[0.06101262103650508]	[0.5552098417411521]
[1 0 0 0 0 0 0 0 0 1]	[513.0]	[152981.0]	[0.10356588454557702]	[0.6587757262867291]
[0 1 0 0 0 0 1 0 0 0]	[264.0]	[85004.0]	[0.05754645642211927]	[0.7163221827088484]
[0 0 0 1 0 0 1 1 0 1]	[77.0]	[-47579.0]	[-0.03221028245856681]	[0.6841119002502816]
[0 1 1 0 1 0 1 1 0 0]	[428.0]	[143716.0]	[0.09729361596114647]	[0.7814055162114281]
[1 0 0 1 1 1 0 0 0 1]	[625.0]	[143125.0]	[0.09689351766288434]	[0.8782990338743124]
[1 1 0 0 1 0 0 0 1 0]	[802.0]	[76396.0]	[0.051718967164183144]	[0.9300180010384955]
[1 1 0 0 0 0 1 0 0 0]	[776.0]	[90124.0]	[0.06101262103650508]	[0.9910306220750006]
[1 1 1 0 0 0 0 0 1 1]	[899.0]	[13249.0]	[0.00896937792499951]	[1.0]

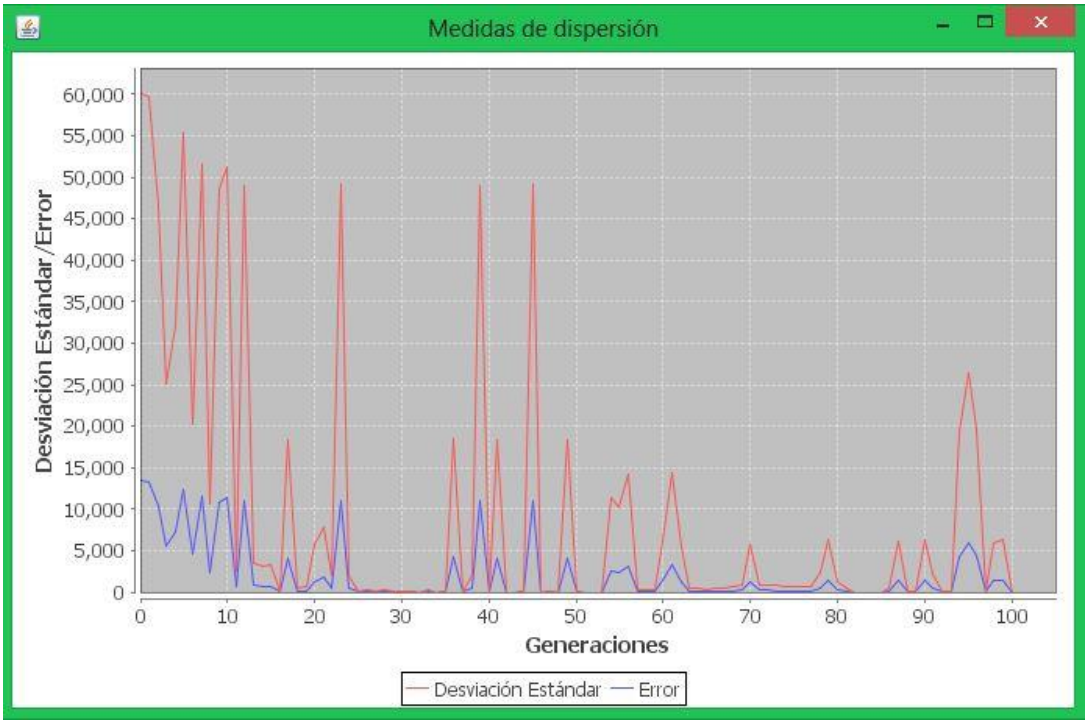
Suma:	1477137.0
Media:	73856.85
Desviación Estándar:	63188.16030339465
Máximo:	152981.0
Mínimo:	-47579.0
Error:	14129.302181154417

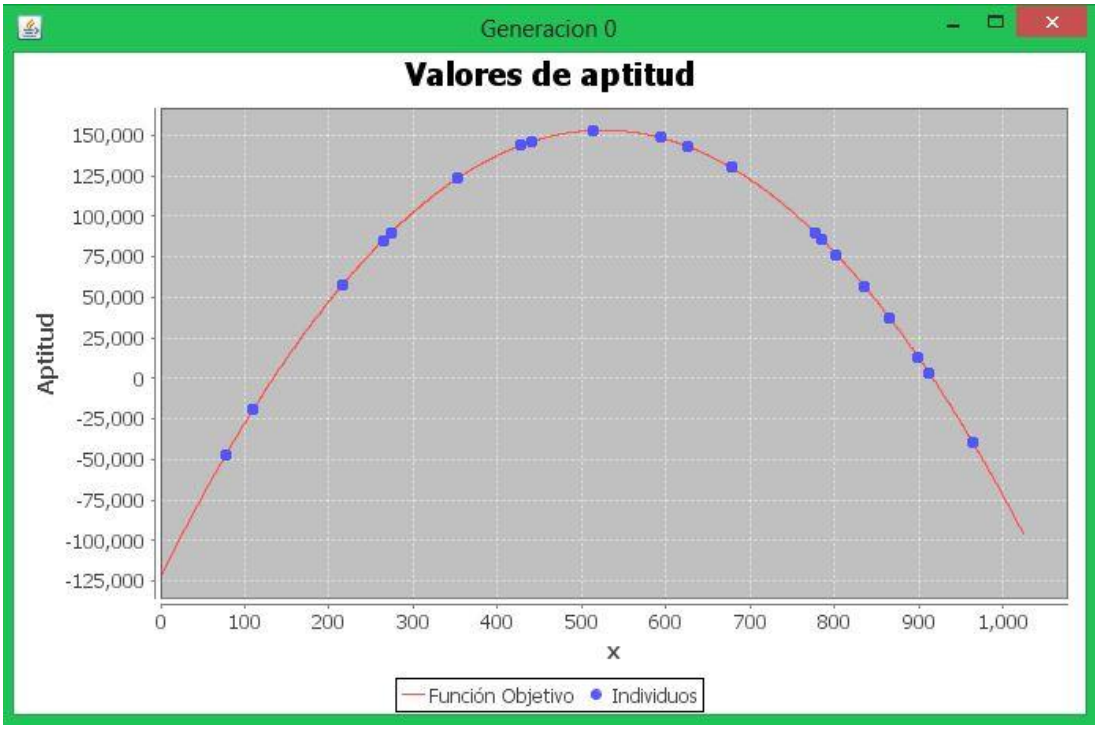
Población Final:

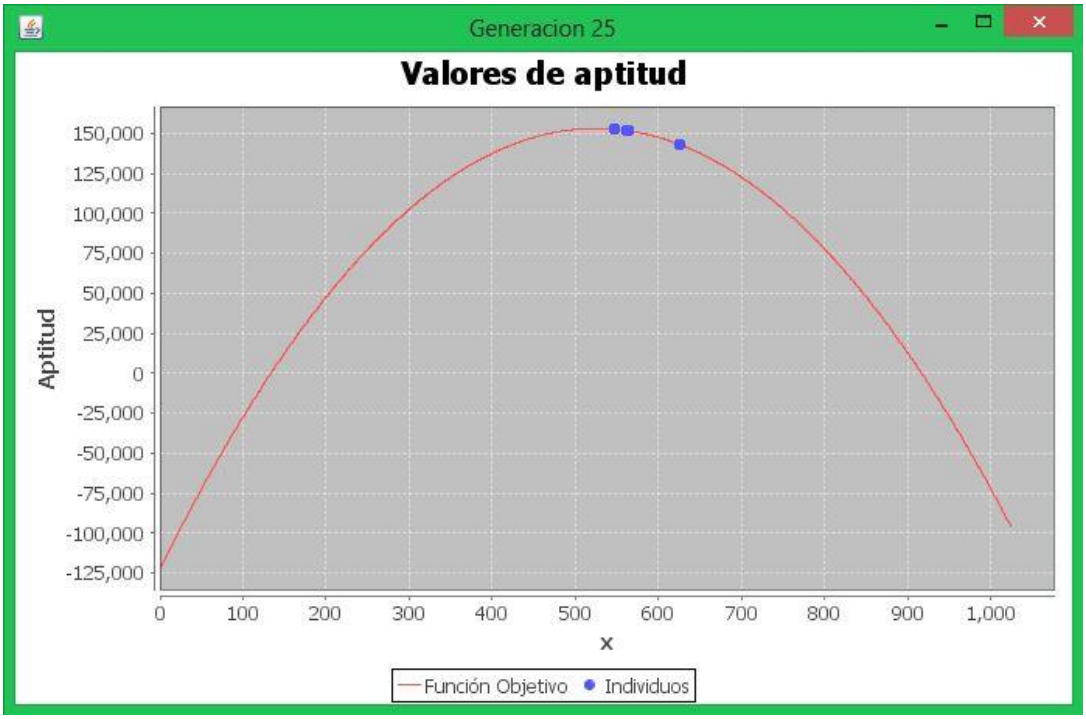
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.05]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.1]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.15000000000000002]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.2]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.25]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.3]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.35]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.39999999999999997]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.44999999999999996]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.49999999999999994]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.5499999999999999]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.6]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.65]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.7000000000000001]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.7500000000000001]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.8000000000000002]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.8500000000000002]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.9000000000000002]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[0.9500000000000003]
[1 0 0 0 1 1 1 0 0 0]	[568.0]	[151276.0]	[0.05]	[1.0000000000000002]

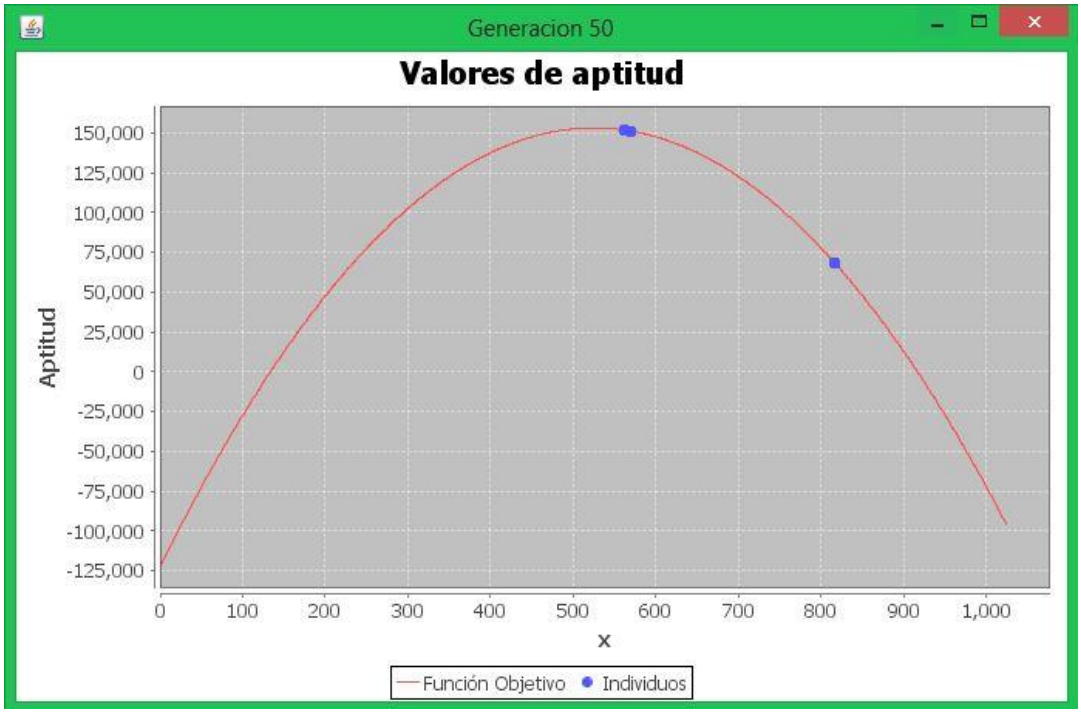
Suma:	3025520.0
Media:	151276.0
Desviación Estándar:	0.0
Máximo:	151276.0
Mínimo:	151276.0
Error:	0.0

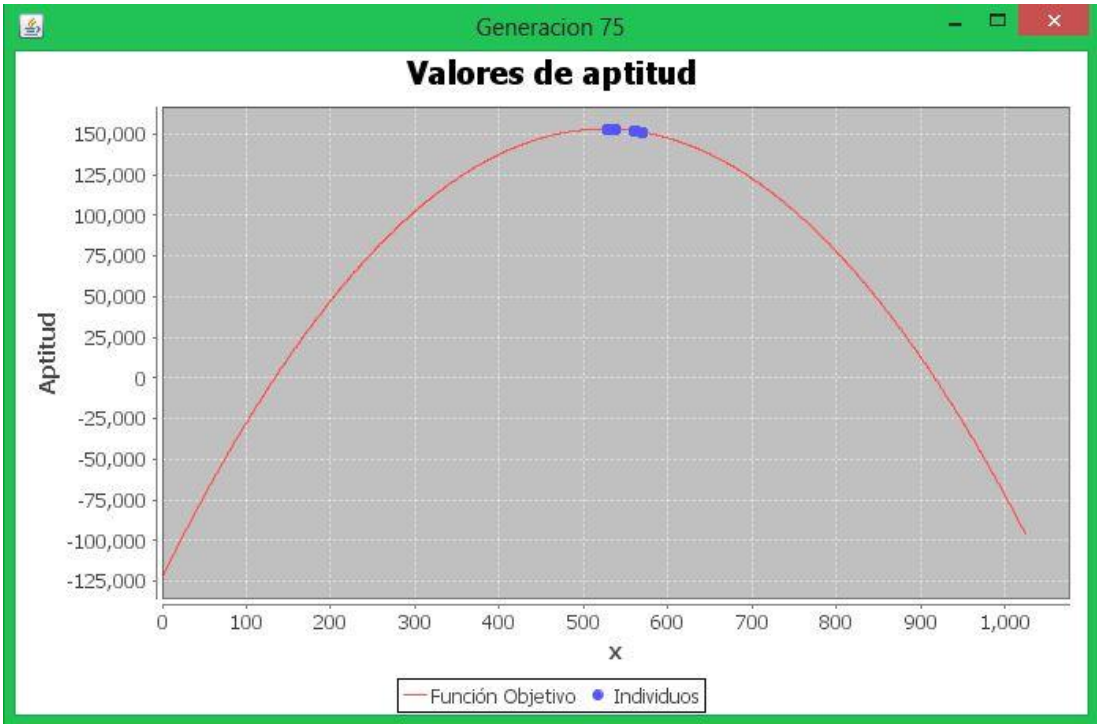

Medidas de dispersión
150,000
125,000
100,000
75,000
50,000
25,000
0
-25,000
-50,000
-75,000
Aptitud
0
10
20
30
40
50
60
70
80
90
100
Generaciones
Promedio
Valor Máximo
Valor Mínimo

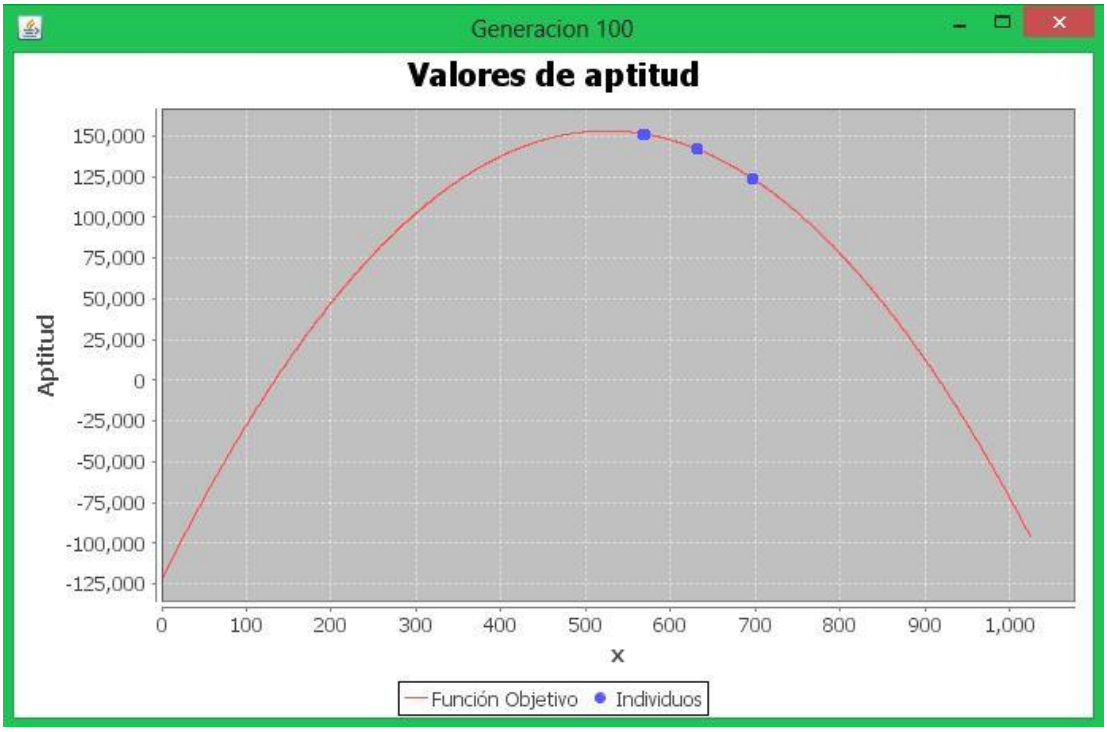

Medidas de dispersión
60,000
55,000
50,000
45,000
40,000
35,000
30,000
25,000
20,000
15,000
10,000
5,000
0
Desviación Estándar /Error
0
10
20
30
40
50
60
70
80
90
100
Generaciones
Desviación Estándar
Error


Generacion 0
Valores de aptitud
Aptitud
150,000
125,000
100,000
75,000
50,000
25,000
0
-25,000
-50,000
-75,000
-100,000
-125,000
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos


Generacion 25
Valores de aptitud
Aptitud
150,000
125,000
100,000
75,000
50,000
25,000
0
-25,000
-50,000
-75,000
-100,000
-125,000
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos


Generacion 50
Valores de aptitud
Aptitud
150,000
125,000
100,000
75,000
50,000
25,000
0
-25,000
-50,000
-75,000
-100,000
-125,000
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos


Generacion 75
Valores de aptitud
Aptitud
150,000
125,000
100,000
75,000
50,000
25,000
0
-25,000
-50,000
-75,000
-100,000
-125,000
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos



Resultados y graficas de la función $f(x) = (x - 350)^2$

Individuos: 20 Genes: 9 Iteraciones: 60

Población Inicial:

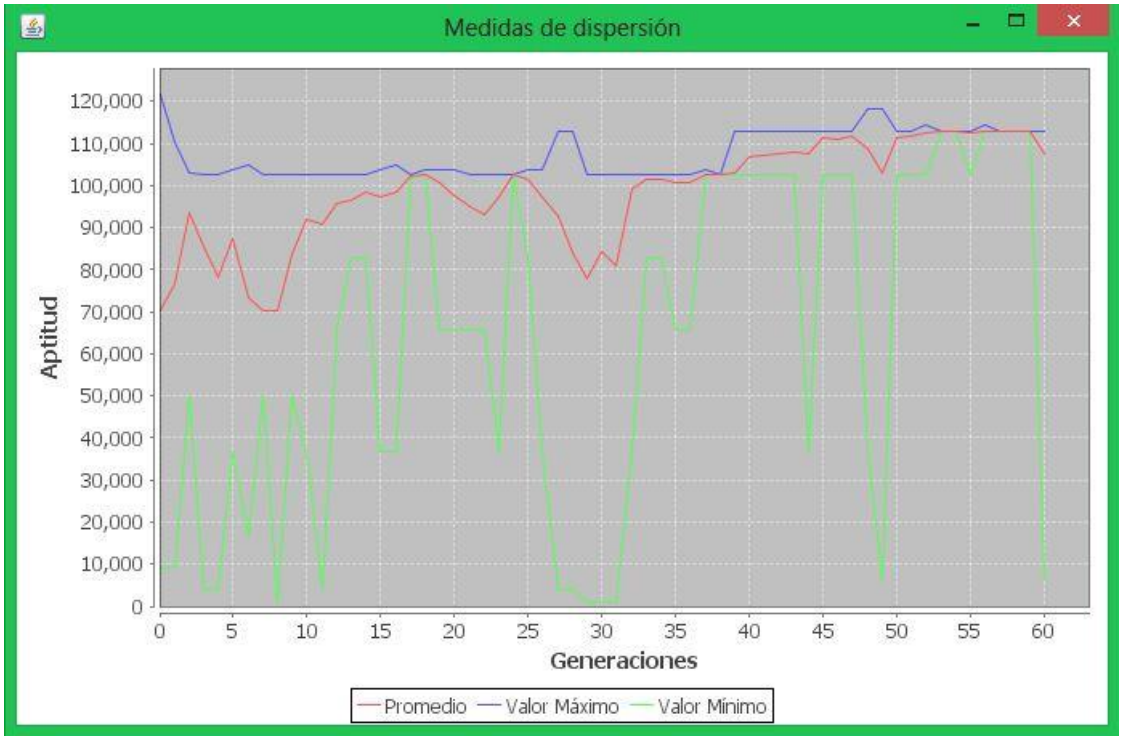
[1 0 1 0 0 1 0 0 1]	[329.0]	[441.0]	[7.453655502521736E-4]	[7.453655502521736E-4]
[1 1 0 0 0 0 0 0 0]	[384.0]	[1156.0]	[0.0019538380410238382]	[0.002699203591276012]
[0 1 0 0 0 1 0 0 0]	[136.0]	[45796.0]	[0.07740308557675406]	[0.08010228916803007]
[0 1 0 1 1 1 1 0 0]	[188.0]	[26244.0]	[0.04435685601092527]	[0.12445914517895534]
[0 0 1 1 1 0 1 1 0]	[118.0]	[53824.0]	[0.090971780899712]	[0.21543092607866735]
[0 1 0 0 0 1 0 0 0]	[136.0]	[45796.0]	[0.07740308557675406]	[0.2928340116554214]
[1 1 0 0 0 1 1 0 1]	[397.0]	[2209.0]	[0.003733588436523926]	[0.2965676000919453]
[0 1 1 0 0 0 0 1 1]	[195.0]	[24025.0]	[0.04060636586124369]	[0.33717396595318905]
[0 1 0 1 0 1 1 0 0]	[172.0]	[31684.0]	[0.053551387968684507]	[0.3907253539218736]
[1 1 1 1 0 1 0 1 1]	[491.0]	[19881.0]	[0.03360229592871534]	[0.4243276498505889]
[0 0 0 0 1 1 1 1 0]	[30.0]	[102400.0]	[0.17307354273429154]	[0.5974011925848804]
[1 1 1 1 1 0 0 1 0]	[498.0]	[21904.0]	[0.03702151250050705]	[0.6344227050853875]
[0 0 1 1 1 1 1 0 1]	[125.0]	[50625.0]	[0.08556492286058115]	[0.7199876279459686]
[0 1 1 1 1 1 0 1 1]	[251.0]	[9801.0]	[0.01656536906580851]	[0.7365529970117771]
[1 0 0 1 1 0 1 0 1]	[309.0]	[1681.0]	[0.002841177981800235]	[0.7393941749935774]
[1 1 0 0 0 1 1 0 1]	[397.0]	[2209.0]	[0.003733588436523926]	[0.7431277634301013]
[1 1 1 0 0 1 1 0 1]	[461.0]	[12321.0]	[0.020824600781535217]	[0.7639523642116366]
[1 1 1 1 0 0 0 1 1]	[483.0]	[17689.0]	[0.029897440404559408]	[0.793849804616196]
[1 0 1 0 1 0 0 0 1]	[337.0]	[169.0]	[2.8563895236421164E-4]	[0.7941354435685603]
[0 0 0 0 0 0 0 0 1]	[1.0]	[121801.0]	[0.2058645564314399]	[1.0000000000000002]

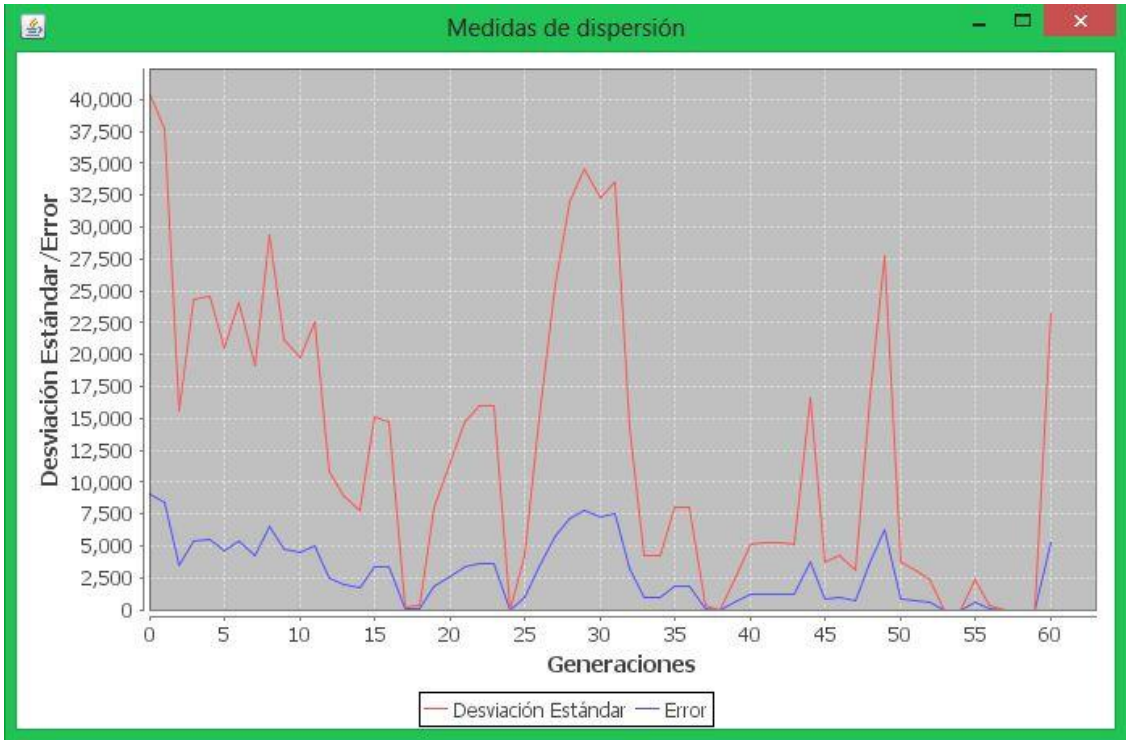
Suma:	591656.0
Media:	29582.8
Desviación Estándar:	32551.98809228094
Máximo:	121801.0
Mínimo:	169.0
Error:	7278.845817710388

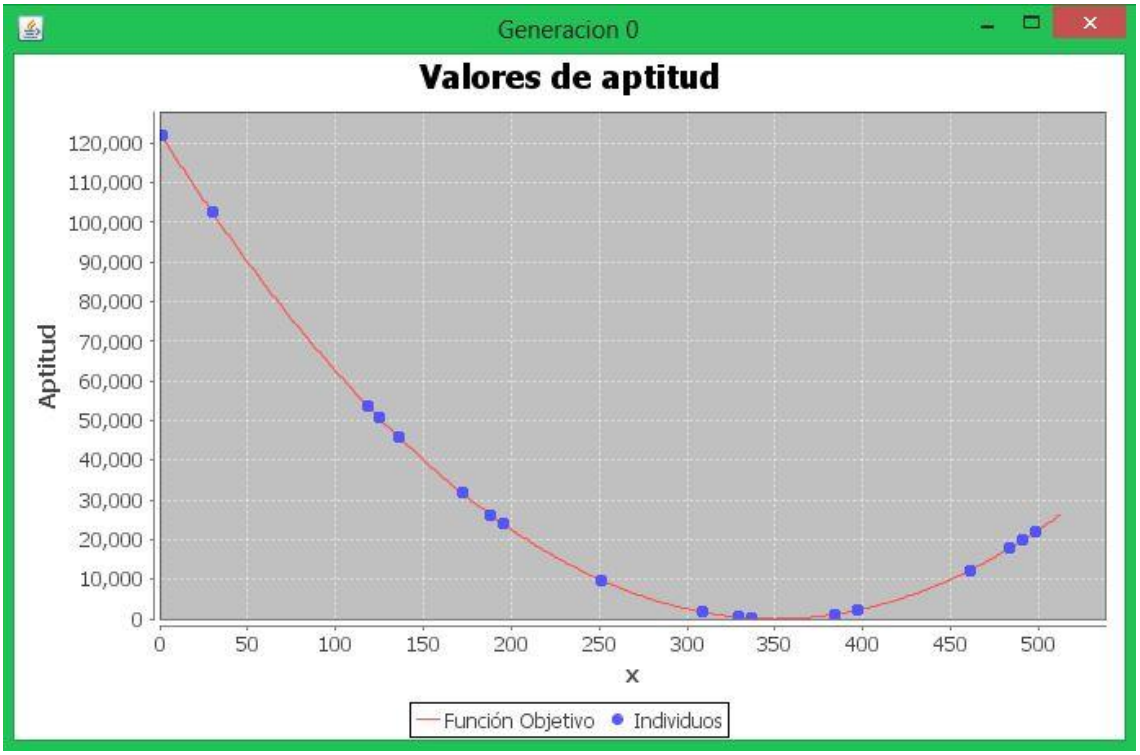
Población Final:

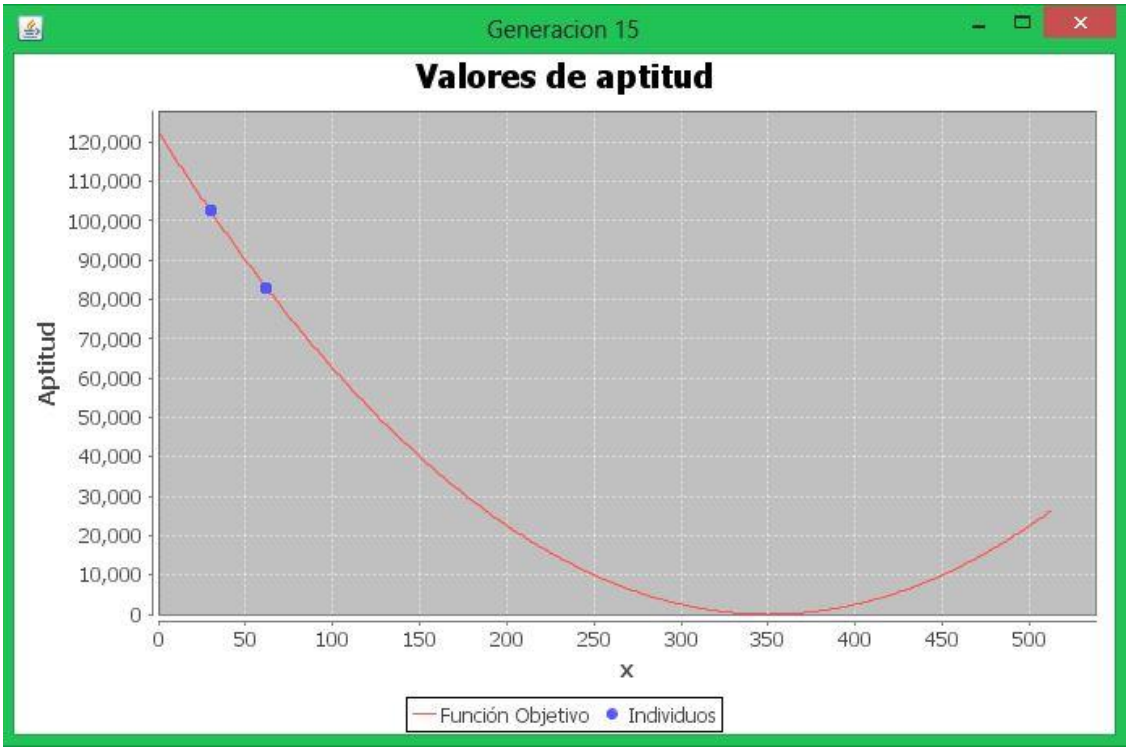
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.05247501189909567]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.10495002379819134]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.157425035697287]
[1 0 0 0 0 1 1 1 0]	[270.0]	[6400.0]	[0.002974773917182294]	[0.16039980961446929]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.21287482151356496]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.2653498334126606]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.31782484531175625]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.3702998572108519]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.42277486910994755]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.4752498810090432]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.5277248929081388]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.5801999048072345]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.6326749167063301]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.6851499286054258]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.7376249405045214]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.7900999524036171]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.8425749643027127]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.8950499762018084]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.947524988100904]
[0 0 0 0 0 1 1 1 0]	[14.0]	[112896.0]	[0.05247501189909567]	[0.9999999999999997]

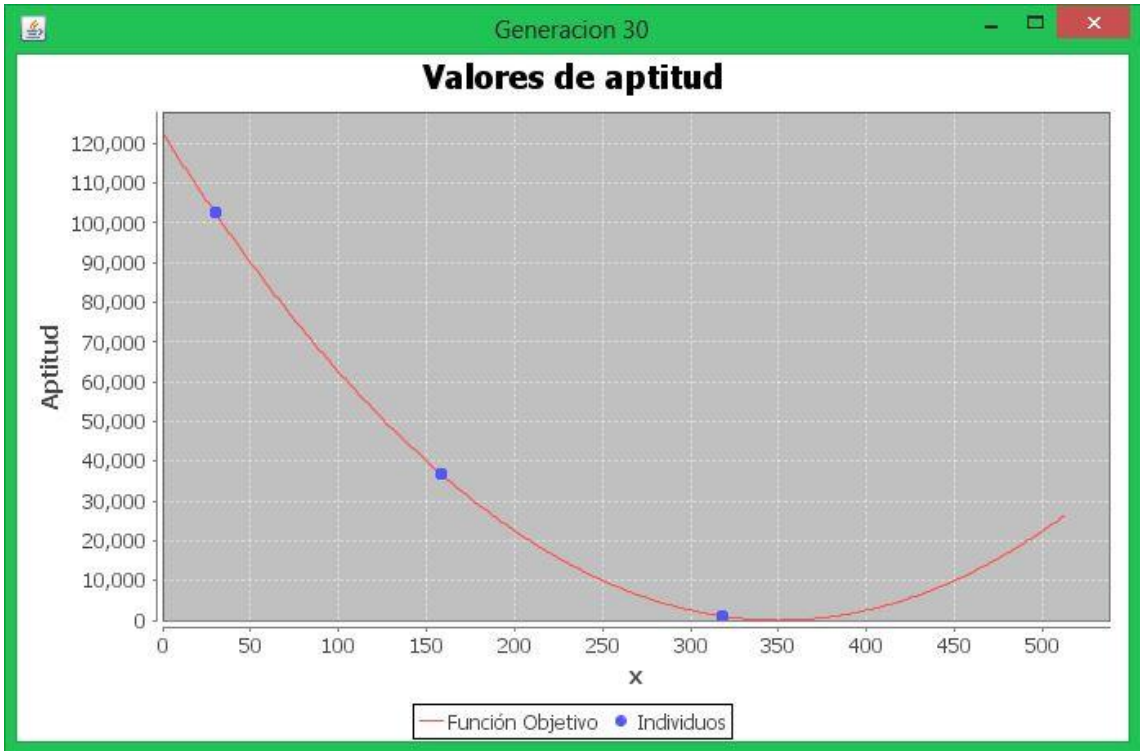
Suma:	2151424.0
Media:	107571.2
Desviación Estándar:	23210.265094565395
Máximo:	112896.0
Mínimo:	6400.0
Error:	5189.973052723881

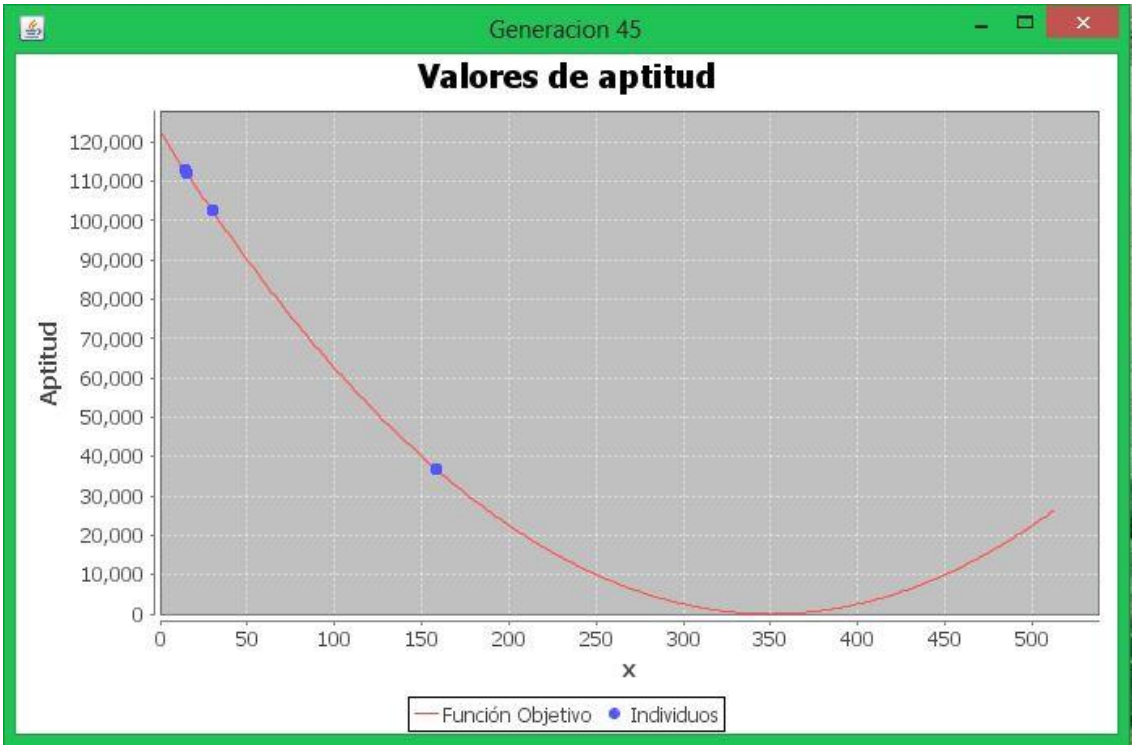


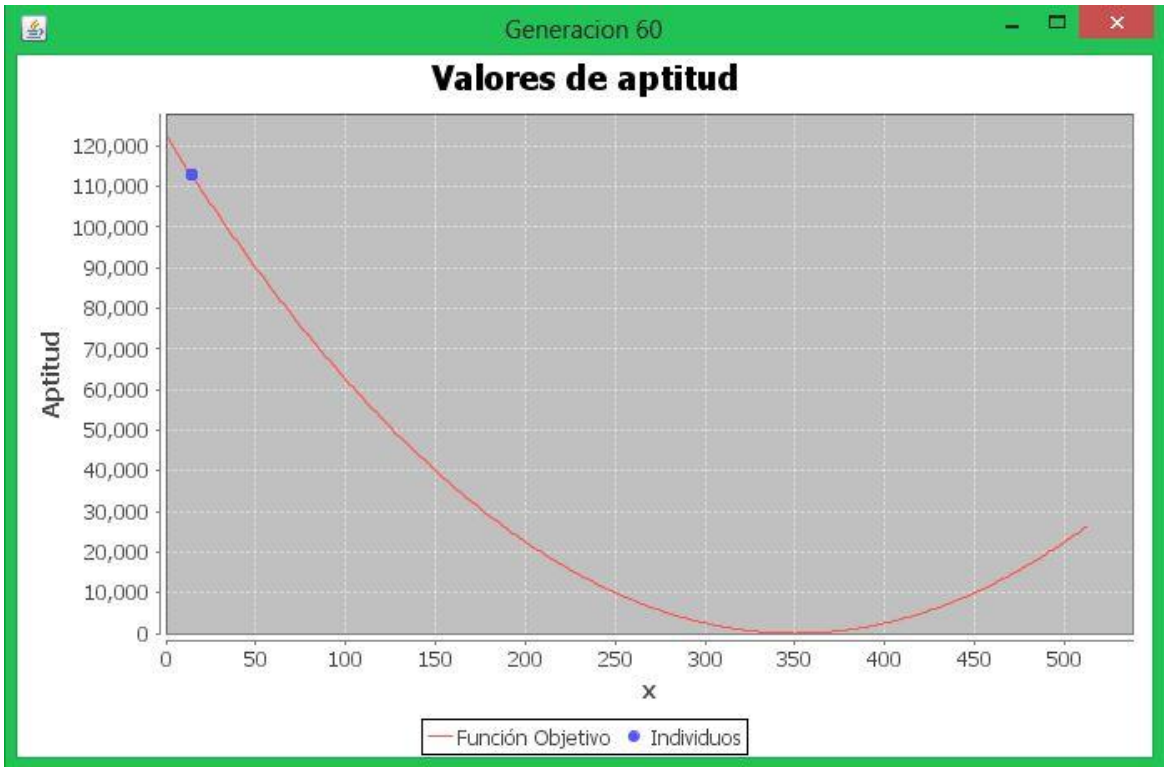

Medidas de dispersión
Desviación Estándar /Error
40,000
37,500
35,000
32,500
30,000
27,500
25,000
22,500
20,000
17,500
15,000
12,500
10,000
7,500
5,000
2,500
0
0
5
10
15
20
25
30
35
40
45
50
55
60
Generaciones
Desviación Estándar
Error


Generacion 0
Valores de aptitud
Aptitud
120,000
110,000
100,000
90,000
80,000
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
0
50
100
150
200
250
300
350
400
450
500
x
Función Objetivo
Individuos


Generacion 15
Valores de aptitud
Aptitud
120,000
110,000
100,000
90,000
80,000
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
0
50
100
150
200
250
300
350
400
450
500
x
Función Objetivo
Individuos


Generacion 30
Valores de aptitud
Aptitud
120,000
110,000
100,000
90,000
80,000
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
0
50
100
150
200
250
300
350
400
450
500
x
Función Objetivo
Individuos


Generacion 45
Valores de aptitud
Aptitud
120,000
110,000
100,000
90,000
80,000
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
0
50
100
150
200
250
300
350
400
450
500
x
Función Objetivo
Individuos


Generacion 60
Valores de aptitud
Aptitud
120,000
110,000
100,000
90,000
80,000
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
0
50
100
150
200
250
300
350
400
450
500
x
Función Objetivo
Individuos

Resultados y graficas de la función $f(x) = 80 - (x-25)^2 e^{-\frac{(x-25)^2}{5}} + 125$

Individuos: 20 Genes: 6 Iteraciones: 60

Población Inicial:

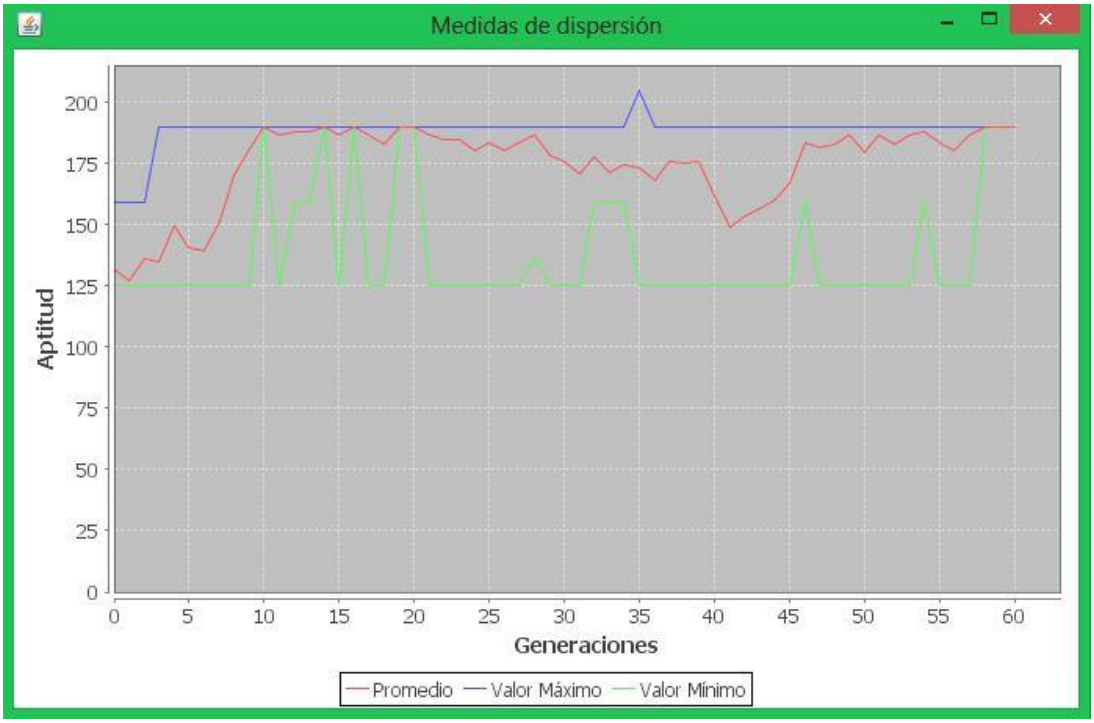
[0 0 1 0 1 1]	[11.0]	[125.0]	[0.04899838475185871]	[0.04899838475185871]
[0 0 0 1 0 1]	[5.0]	[125.0]	[0.04899838475185871]	[0.09799676950371743]
[0 1 0 0 1 0]	[18.0]	[125.0017189995824]	[0.04899905857748212]	[0.14699582808119954]
[0 0 1 1 1 1]	[15.0]	[124.99999995877693]	[0.0489983847356998]	[0.19599421281689933]
[1 1 1 0 0 0]	[56.0]	[125.0]	[0.04899838475185871]	[0.24499259756875805]
[0 1 1 1 0 0]	[28.0]	[136.73622106373264]	[0.05359883175356783]	[0.29859142932232585]
[1 1 1 0 1 0]	[58.0]	[125.0]	[0.04899838475185871]	[0.34758981407418454]
[0 0 1 0 1 1]	[11.0]	[125.0]	[0.04899838475185871]	[0.39658819882604324]
[0 1 0 1 0 1]	[21.0]	[127.60878105461543]	[0.05002099321463797]	[0.4466091920406812]
[0 1 0 1 1 1]	[23.0]	[159.14900127290883]	[0.062384351977952314]	[0.5089935440186335]
[0 0 0 0 1 1]	[3.0]	[125.0]	[0.04899838475185871]	[0.5579919287704922]
[1 0 1 1 0 1]	[45.0]	[125.0]	[0.04899838475185871]	[0.6069903135223509]
[1 0 1 1 1 1]	[47.0]	[125.0]	[0.04899838475185871]	[0.6559886982742096]
[0 0 1 0 0 0]	[8.0]	[125.0]	[0.04899838475185871]	[0.7049870830260683]
[0 1 0 1 0 1]	[21.0]	[127.60878105461543]	[0.05002099321463797]	[0.7550080762407062]
[1 1 0 0 0 0]	[48.0]	[125.0]	[0.04899838475185871]	[0.8040064609925649]
[1 1 0 1 1 0]	[54.0]	[125.0]	[0.04899838475185871]	[0.8530048457444236]
[1 1 0 0 1 1]	[51.0]	[125.0]	[0.04899838475185871]	[0.9020032304962823]
[1 1 1 1 1 1]	[63.0]	[125.0]	[0.04899838475185871]	[0.951001615248141]
[0 0 0 1 0 0]	[4.0]	[125.0]	[0.04899838475185871]	[0.9999999999999997]

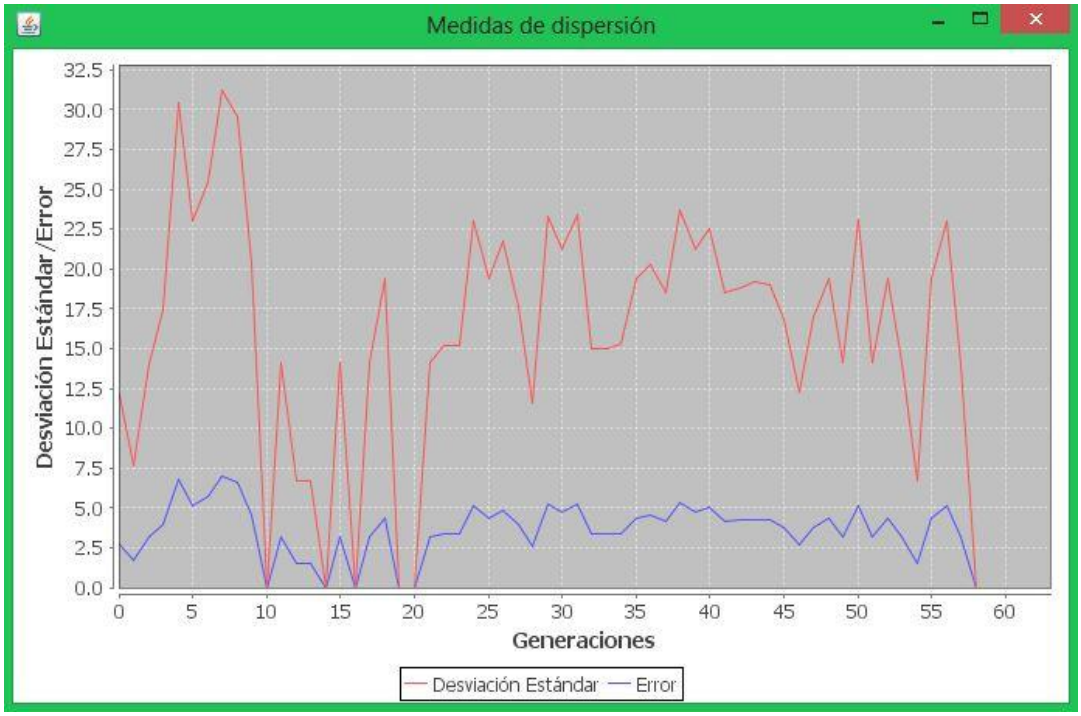
Suma:	2551.1045034042318
Media:	127.55522517021159
Desviación Estándar:	7.7036392033930134
Máximo:	159.14900127290883
Mínimo:	124.99999995877693
Error:	1.7225860932919106

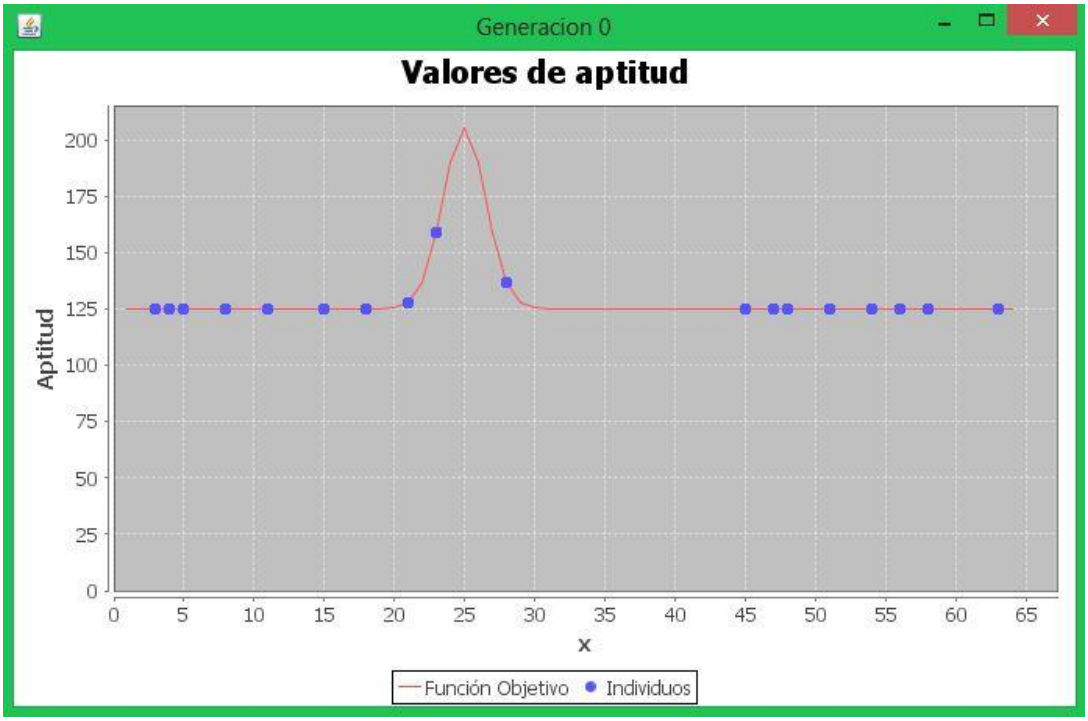
Población Final:

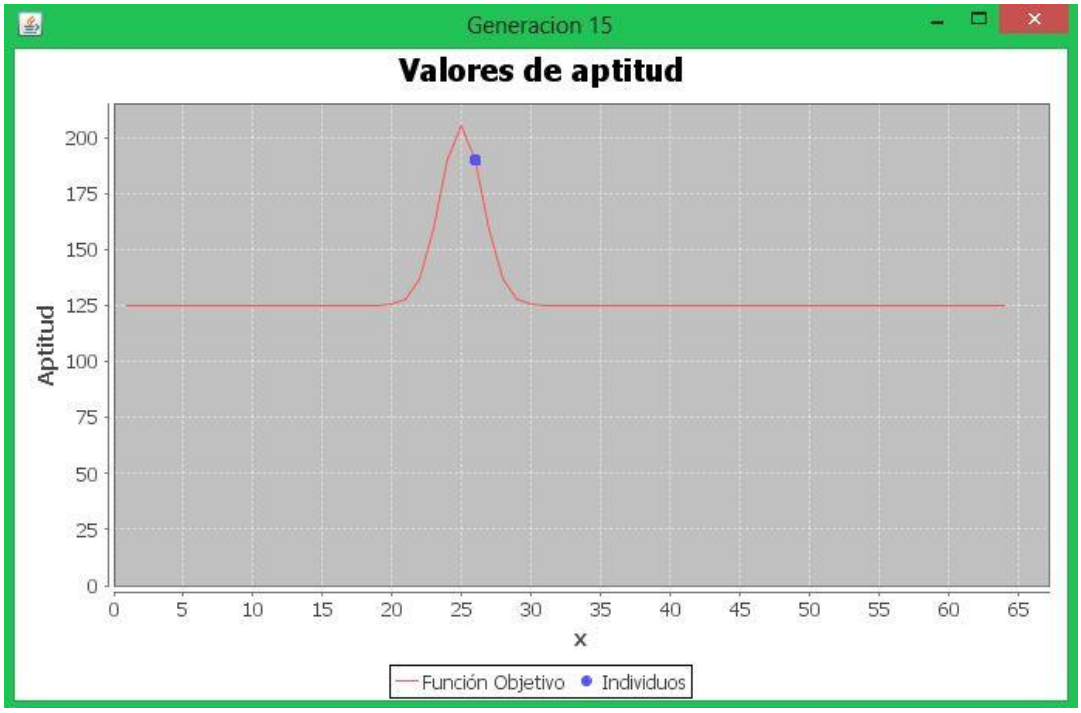
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.04999999999999998]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.09999999999999996]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.14999999999999994]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.19999999999999993]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.24999999999999992]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.2999999999999999]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.34999999999999987]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.39999999999999986]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.44999999999999984]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.49999999999999983]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.5499999999999998]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.5999999999999998]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.6499999999999997]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.6999999999999996]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.7499999999999996]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.7999999999999995]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.8499999999999994]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.8999999999999994]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.9499999999999993]
[0 1 1 0 1 0]	[26.0]	[189.67972949316055]	[0.04999999999999998]	[0.9999999999999992]

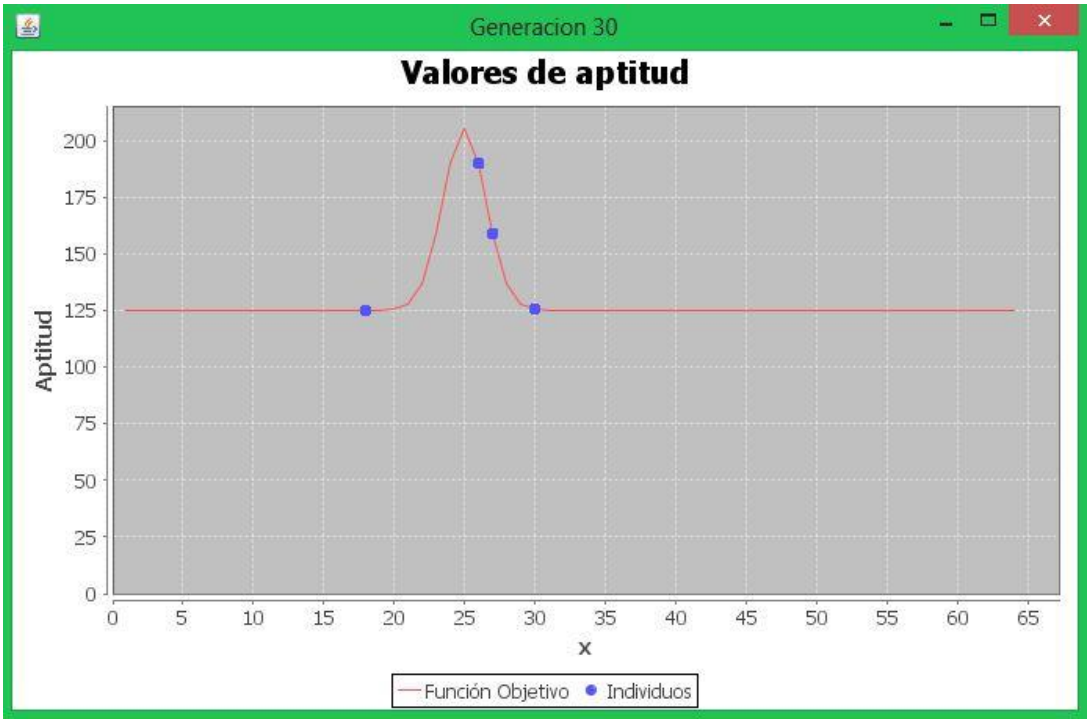
Suma:	3793.594589863212
Media:	189.6797294931606
Desviación Estándar:	5.6843418860808015E-14
Máximo:	189.67972949316055
Mínimo:	189.67972949316055
Error:	1.2710574864626037E-14

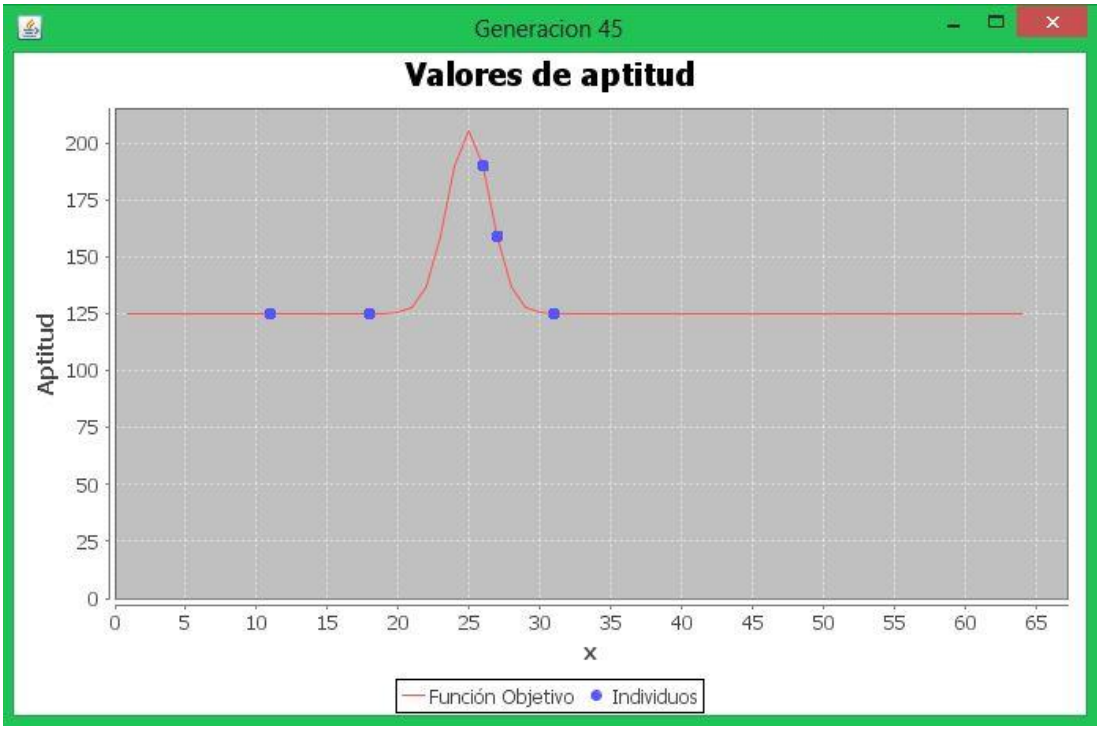

Medidas de dispersión
Aptitud
Generaciones
Promedio
Valor Máximo
Valor Mínimo

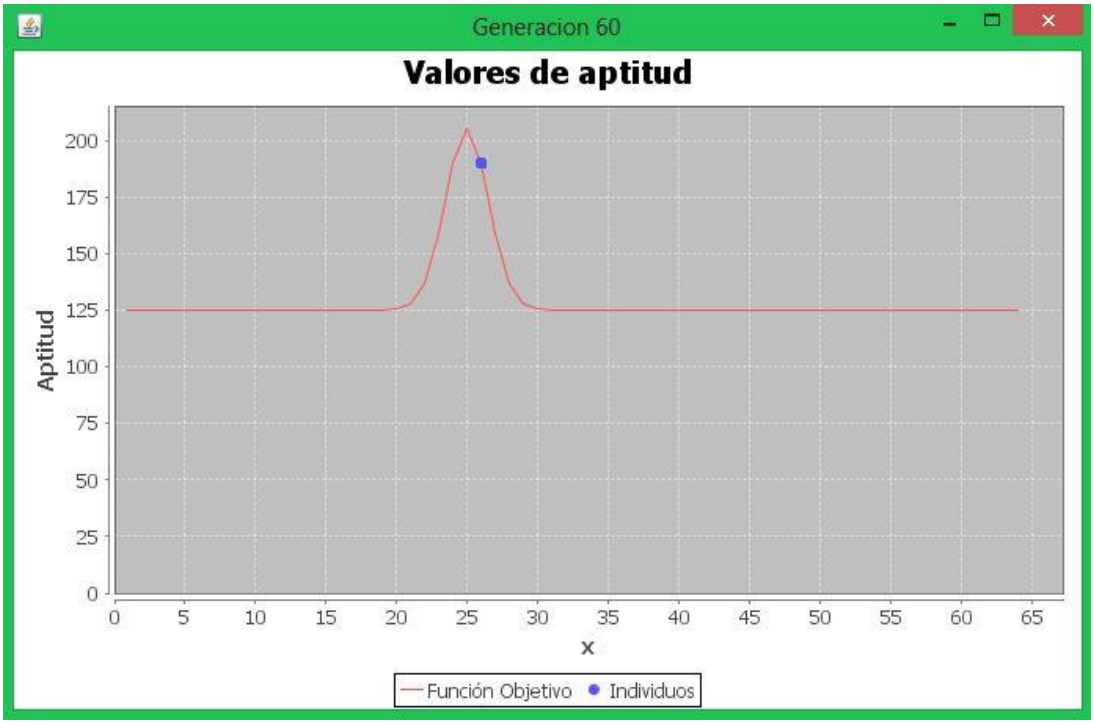

Medidas de dispersión
Desviación Estándar /Error
Generaciones
Desviación Estándar
Error


Generacion 0
Valores de aptitud
Aptitud
x
Función Objetivo
Individuos


Generacion 15
Valores de aptitud
Aptitud
x
Función Objetivo
Individuos


Generacion 30
Valores de aptitud
Aptitud
x
Función Objetivo
Individuos


Generacion 45
Valores de aptitud
Aptitud
x
Función Objetivo
Individuos



Resultados y graficas de la función $f(x) = (x - 1000)^2 \sin x$

Individuos: 20 Genes: 10 Iteraciones: 100

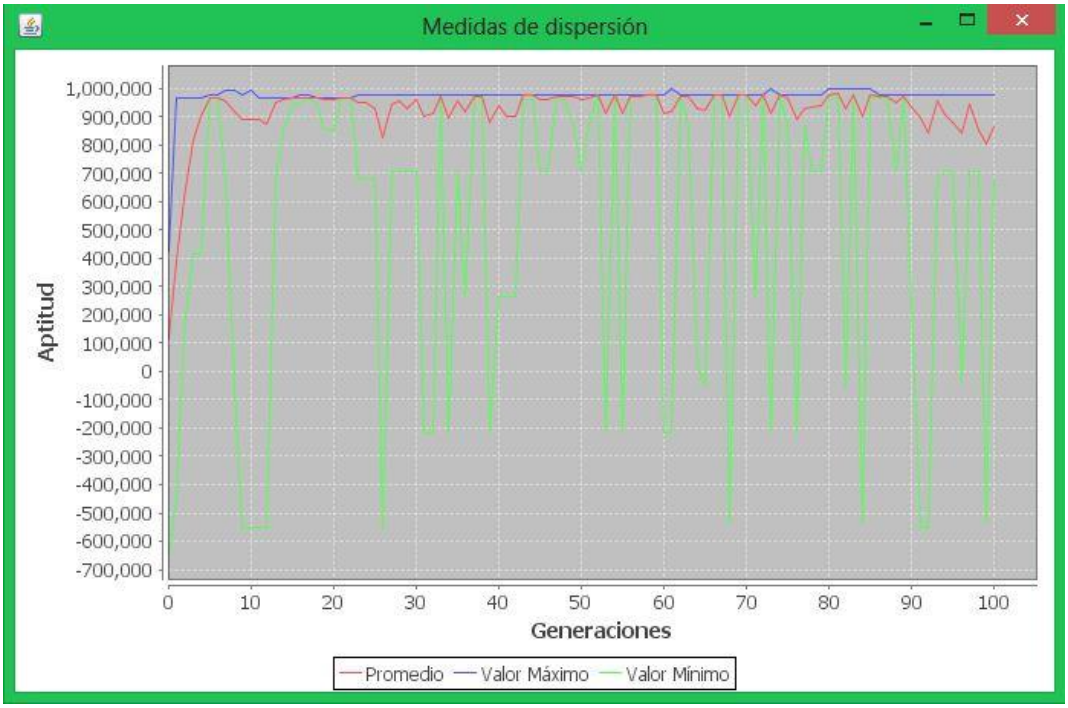
Población Inicial:

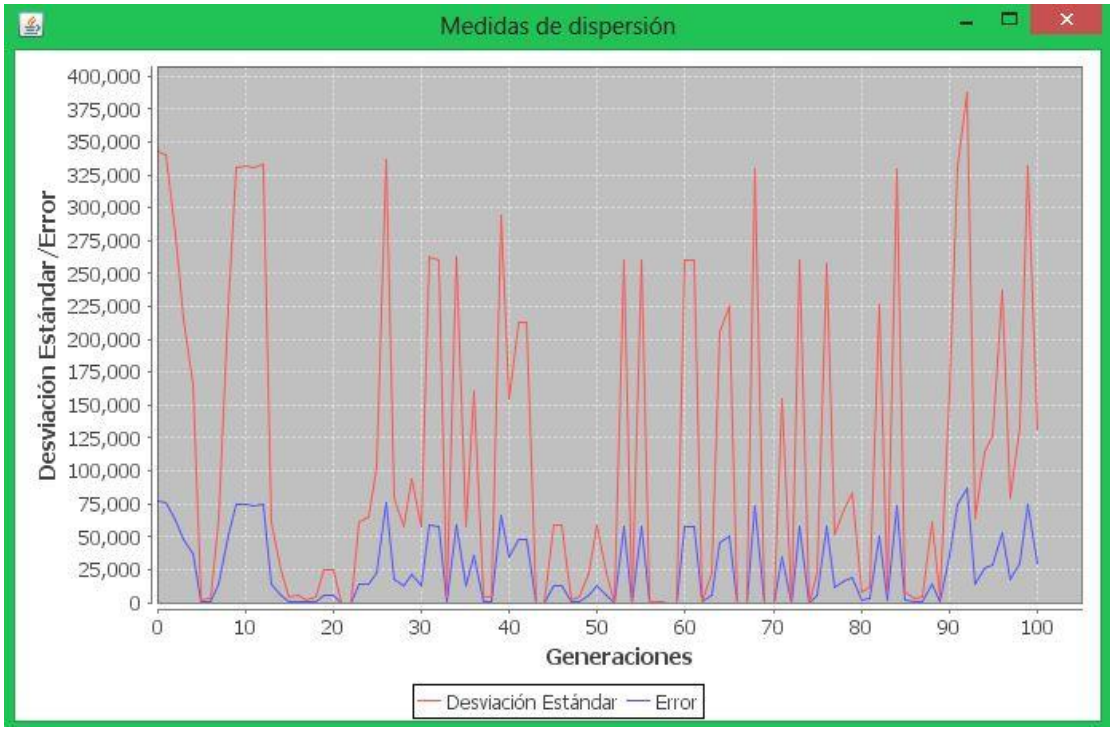
```
[ 0 1 0 1 0 1 0 0 1 1 ]    [ 339.0 ]    [ 407900.8935255838 ]   [ 0.1549825556934649 ]  [ 0.1549825556934649 ]
[ 0 0 0 0 1 1 1 0 0 0 ]    [ 56.0 ]     [ 498316.9272273074 ]   [ 0.1893362631777577 ]  [ 0.3443188188712226 ]
[ 0 0 0 0 1 1 1 1 1 0 ]    [ 62.0 ]     [ 413061.73768778937 ]  [ 0.15694342616590026 ]         [ 0.5012622450371229 ]
[ 1 0 1 0 1 1 1 1 1 1 ]    [ 703.0 ]    [ 84354.68621874336 ]   [ 0.032050689425815564 ]        [ 0.5333129344629385 ]
[ 0 0 0 0 0 1 1 1 1 0 ]    [ 30.0 ]     [ 814843.3024207784 ]   [ 0.3096009337154207 ]  [ 0.8429138681783592 ]
[ 1 1 0 1 1 1 0 1 1 1 ]    [ 887.0 ]    [ -12441.731357242672 ]         [ -0.004727254471866092 ]       [ 0.8381866137064932 ]
[ 1 0 1 0 1 1 1 1 0 1 ]    [ 701.0 ]    [ 84530.30617715453 ]   [ 0.03211741649216271 ]         [ 0.8703040301986559 ]
[ 1 0 0 1 1 0 0 1 1 0 ]    [ 614.0 ]    [ -41068.863467309726 ]         [ -0.015604176211961205 ]       [ 0.8546998539866947 ]
[ 1 0 0 0 0 0 0 1 0 0 ]    [ 516.0 ]    [ -214003.50472552504 ]         [ -0.08131095228316806 ]        [ 0.7733889017035266 ]
[ 1 1 0 1 1 1 1 0 1 0 ]    [ 890.0 ]    [ -11916.173811447714 ]         [ -0.004527568094846232 ]       [ 0.7688613336086804 ]
[ 0 1 0 0 0 1 1 1 1 0 ]    [ 286.0 ]    [ 140518.82144608302 ]  [ 0.053390336761745004 ]        [ 0.8222516703704253 ]
[ 0 1 0 0 1 0 1 0 1 0 ]    [ 298.0 ]    [ 231357.46402713793 ]  [ 0.0879046151229766 ]  [ 0.910156285493402 ]
[ 0 1 1 1 0 0 1 0 1 1 ]    [ 459.0 ]    [ -45785.39566243946 ]  [ -0.017396229686749358 ]       [ 0.8927600558066526 ]
[ 1 1 0 0 0 0 0 1 0 0 ]    [ 772.0 ]    [ 32004.546133329026 ]  [ 0.012160175258948356 ]        [ 0.9049202310656009 ]
[ 1 0 1 1 0 1 1 1 0 0 ]    [ 732.0 ]    [ 70254.47327510487 ]   [ 0.026693292390130706 ]        [ 0.9316135234557317 ]
[ 1 0 1 1 1 1 0 0 0 1 ]    [ 753.0 ]    [ 51166.4526797824 ]    [ 0.019440770363460955 ]        [ 0.9510542938191926 ]
[ 0 1 0 1 0 0 1 0 1 1 ]    [ 331.0 ]    [ 391445.67074701516 ]  [ 0.1487303691422532 ]  [ 1.0997846629614458 ]
[ 1 1 1 1 0 1 1 0 1 1 ]    [ 987.0 ]    [ -8.84477660505737 ]   [ -3.360586124098752E-6 ]       [ 1.0997813023753218 ]
[ 0 1 1 0 1 1 1 0 0 0 ]    [ 440.0 ]    [ 54456.06851634938 ]   [ 0.020690664829722694 ]        [ 1.1204719672050445 ]
[ 0 0 1 1 1 0 1 1 0 1 ]    [ 237.0 ]    [ -317071.9623756632 ]  [ -0.12047196720504437 ]        [ 1.0000000000000002 ]
      Suma:                  2631914.873905926
      Media:                 131595.7436952963
      Desviación Estándar:   255505.54310908972
      Máximo:                814843.3024207784
      Mínimo:                -317071.9623756632
      Error:                 57132.77630199275
```

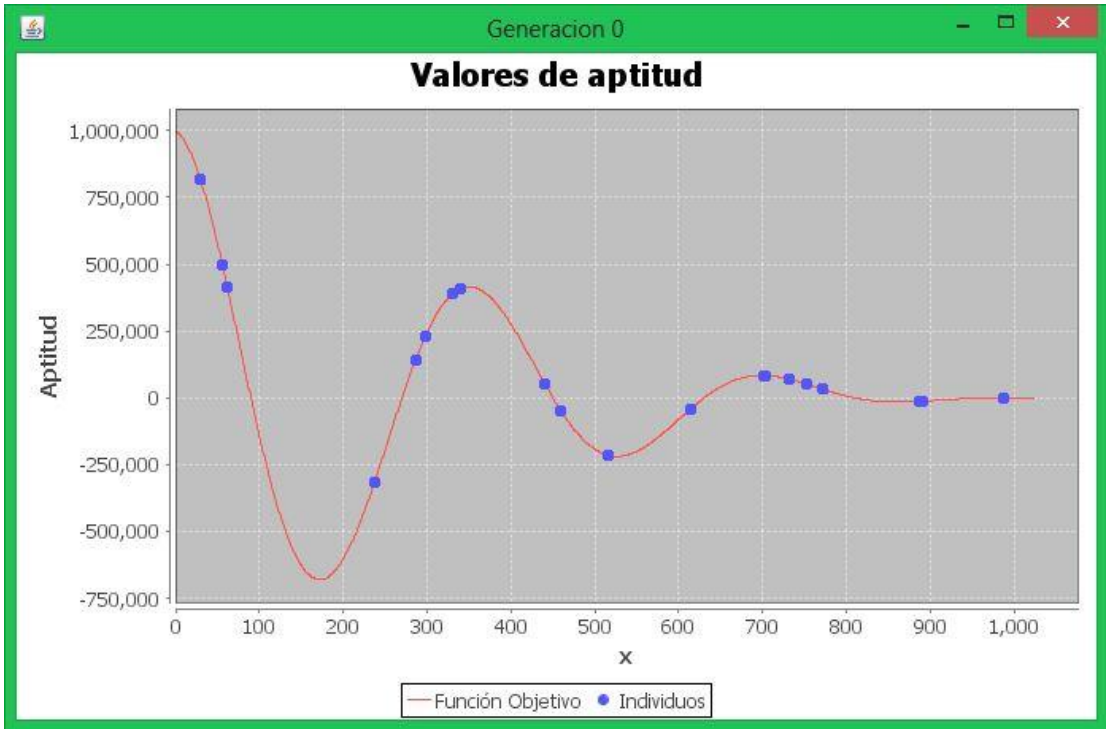
Población Final:

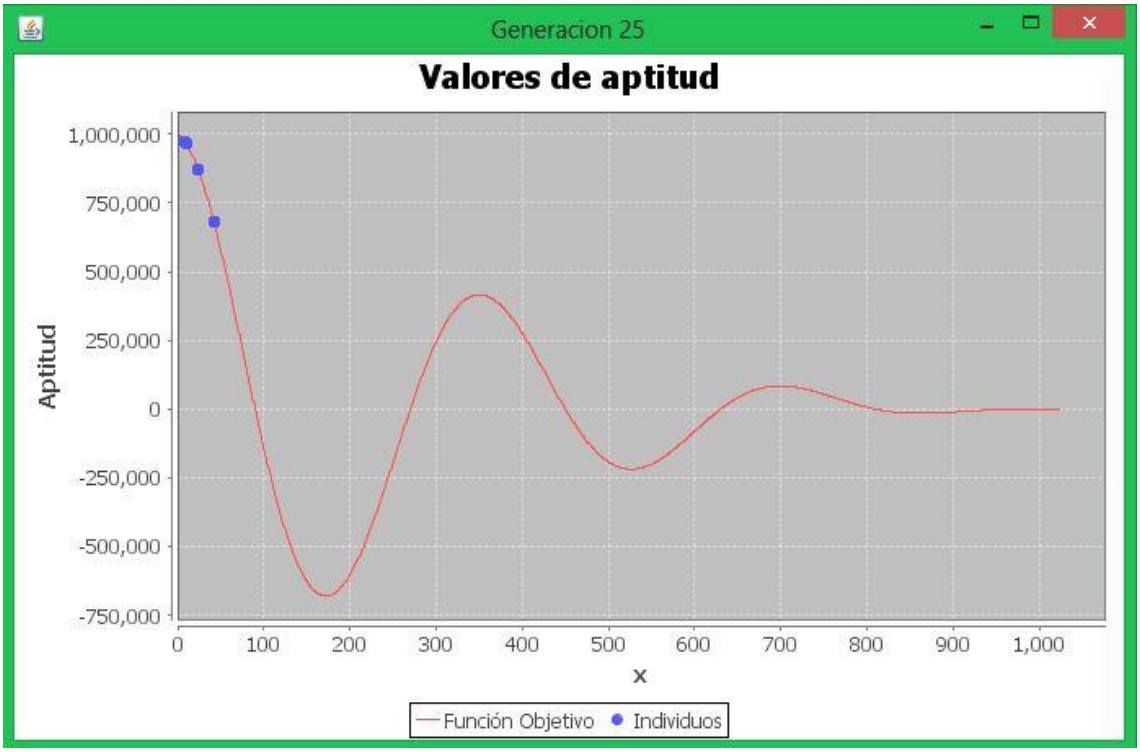
[0 0 0 0 0 0 1 0 0 1]	[9.0]	[969989.9532200135]	[0.05617626268464365]	[0.05617626268464365]
[0 0 0 0 1 0 1 0 0 0]	[40.0]	[705986.5587784501]	[0.0408866981004392]	[0.09706296078508285]
[0 0 0 0 1 0 1 0 1 0]	[42.0]	[682031.5676094353]	[0.03949936220891614]	[0.13656232299399898]
[0 0 0 0 1 0 1 0 0 0]	[40.0]	[705986.5587784501]	[0.0408866981004392]	[0.17744902109443816]
[0 0 0 0 1 0 1 0 0 0]	[40.0]	[705986.5587784501]	[0.0408866981004392]	[0.21833571919487738]
[0 0 0 0 0 0 1 0 0 0]	[8.0]	[974487.1567981046]	[0.056436714959134236]	[0.2747724341540116]
[0 0 0 0 0 0 1 0 0 0]	[8.0]	[974487.1567981046]	[0.056436714959134236]	[0.3312091491131458]
[0 0 0 0 0 0 1 0 1 0]	[10.0]	[965210.0787272651]	[0.05589943972971212]	[0.3871085888428579]
[0 0 0 0 0 0 1 0 0 0]	[8.0]	[974487.1567981046]	[0.056436714959134236]	[0.44354530380199214]
[0 0 0 0 0 0 1 0 1 0]	[10.0]	[965210.0787272651]	[0.05589943972971212]	[0.49944474353170426]
[0 0 0 0 0 0 1 0 0 0]	[8.0]	[974487.1567981046]	[0.056436714959134236]	[0.5558814584908385]
[0 0 0 0 0 0 1 0 1 0]	[10.0]	[965210.0787272651]	[0.05589943972971212]	[0.6117808982205506]
[0 0 0 0 1 0 1 0 0 0]	[40.0]	[705986.5587784501]	[0.0408866981004392]	[0.6526675963209898]
[0 0 0 0 0 0 1 0 1 0]	[10.0]	[965210.0787272651]	[0.05589943972971212]	[0.7085670360507019]
[0 0 0 0 1 0 1 0 0 0]	[40.0]	[705986.5587784501]	[0.0408866981004392]	[0.7494537341511411]
[0 0 0 0 0 0 1 0 0 0]	[8.0]	[974487.1567981046]	[0.056436714959134236]	[0.8058904491102754]
[0 0 0 0 1 0 1 0 0 0]	[40.0]	[705986.5587784501]	[0.0408866981004392]	[0.8467771472107146]
[0 0 0 0 0 0 1 0 0 0]	[8.0]	[974487.1567981046]	[0.056436714959134236]	[0.9032138621698489]
[0 0 0 0 1 0 1 0 0 0]	[40.0]	[705986.5587784501]	[0.0408866981004392]	[0.9441005602702881]
[0 0 0 0 0 0 1 0 1 0]	[10.0]	[965210.0787272651]	[0.05589943972971212]	[1.0000000000000002]

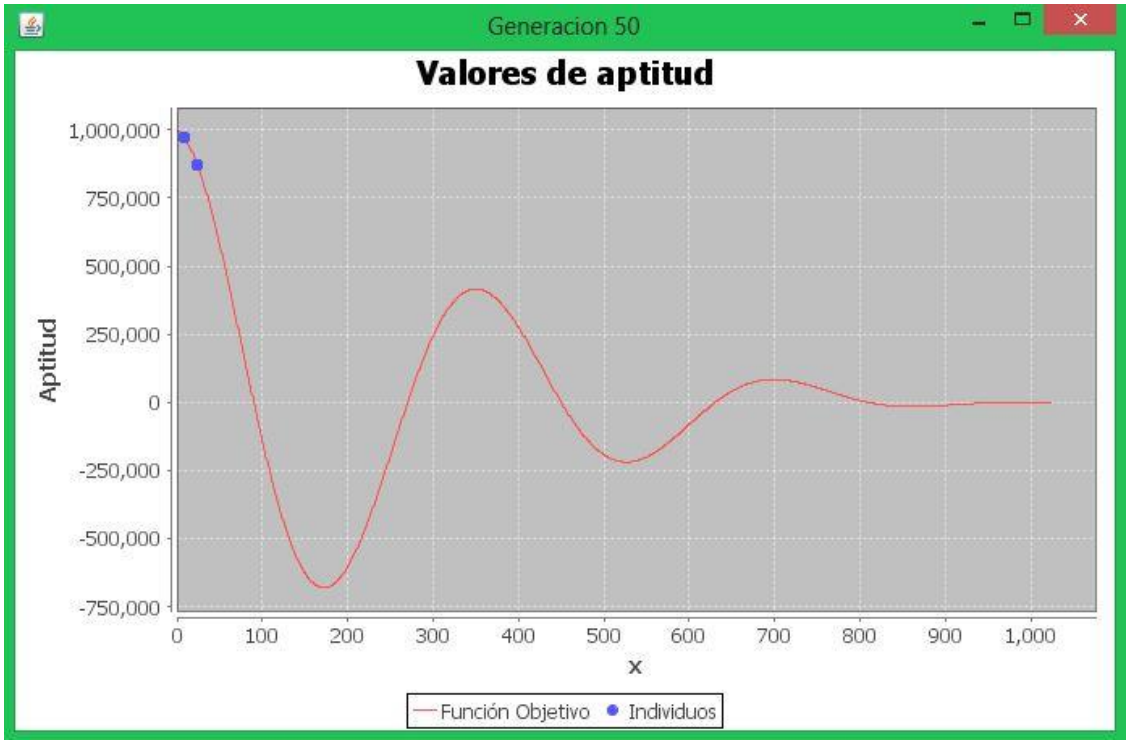
Suma:	1.7266900766703555E7
Media:	863345.0383351775
Desviación Estándar:	131068.18939109819
Máximo:	974487.1567981046
Mínimo:	682031.5676094353
Error:	29307.73811663123

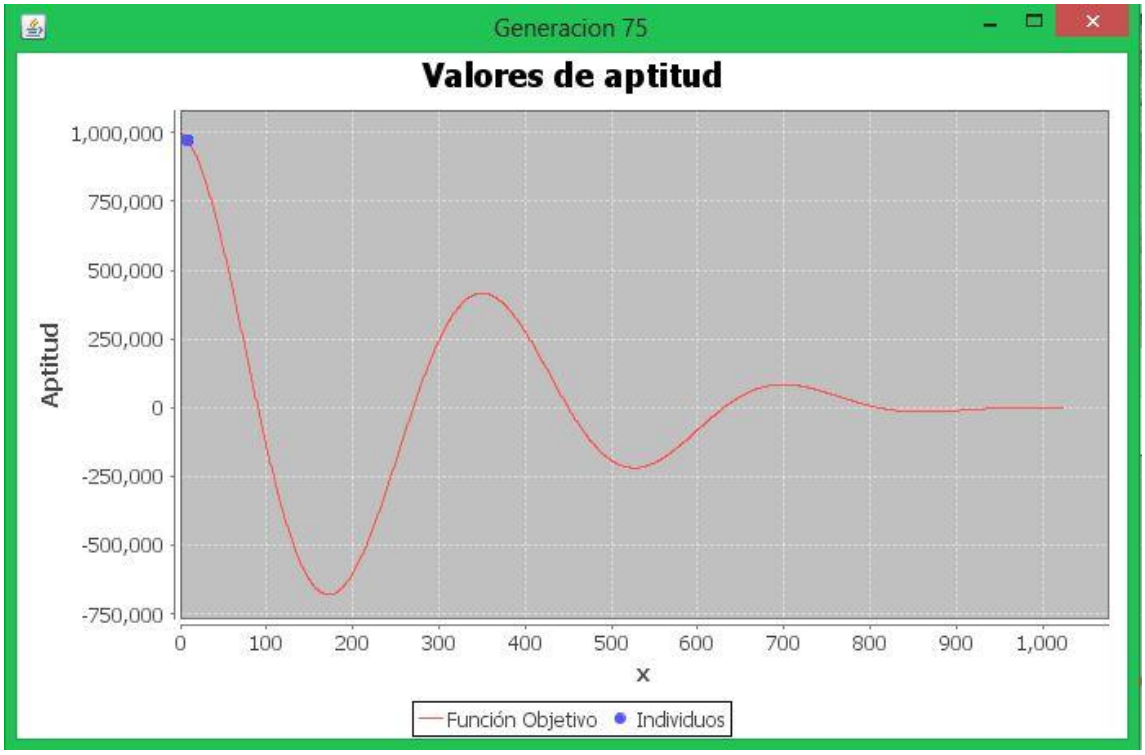


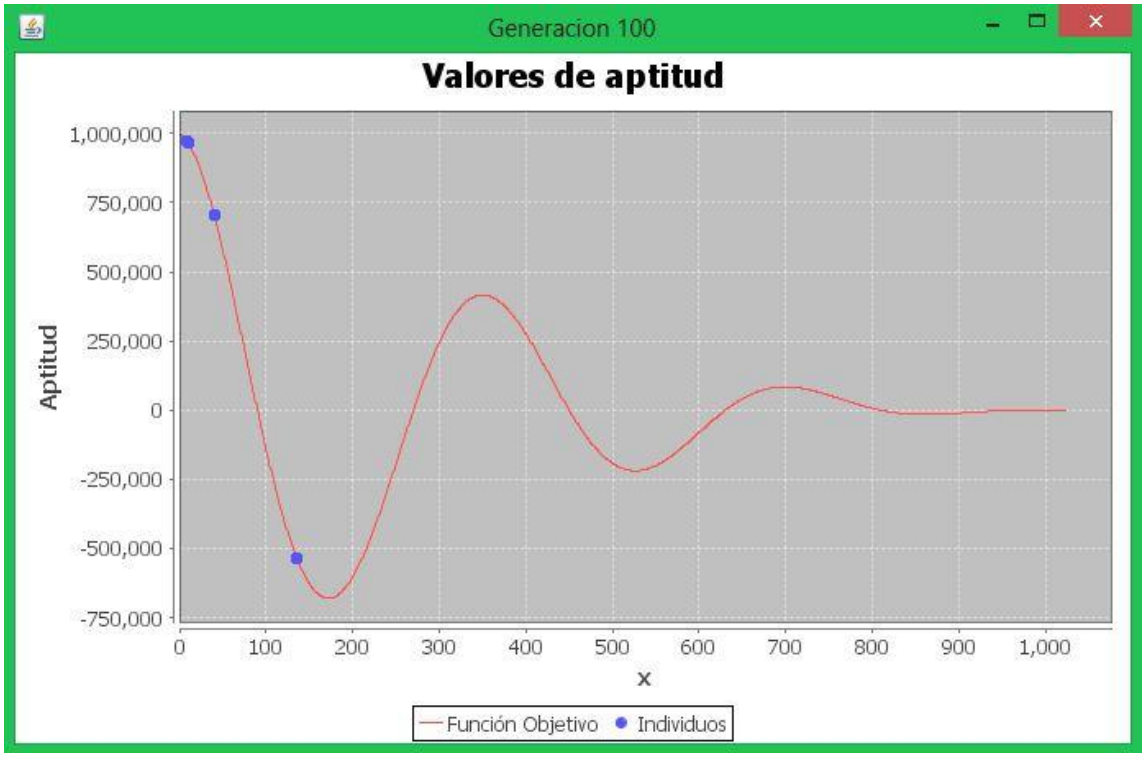

Medidas de dispersión
Desviación Estándar /Error
400,000
375,000
350,000
325,000
300,000
275,000
250,000
225,000
200,000
175,000
150,000
125,000
100,000
75,000
50,000
25,000
0
0
10
20
30
40
50
60
70
80
90
100
Generaciones
Desviación Estándar
Error


Generacion 0
Valores de aptitud
Aptitud
1,000,000
750,000
500,000
250,000
0
-250,000
-500,000
-750,000
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos


Generacion 25
Valores de aptitud
1,000,000
750,000
500,000
250,000
0
-250,000
-500,000
-750,000
Aptitud
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos


Generacion 50
Valores de aptitud
1,000,000
750,000
500,000
250,000
0
-250,000
-500,000
-750,000
Aptitud
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos


Generacion 75
Valores de aptitud
1,000,000
750,000
500,000
250,000
0
-250,000
-500,000
-750,000
Aptitud
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos


Generacion 100
Valores de aptitud
1,000,000
750,000
500,000
250,000
0
-250,000
-500,000
-750,000
Aptitud
0
100
200
300
400
500
600
700
800
900
1,000
x
Función Objetivo
Individuos

3.5.2 Código utilizado para la resolución del problema de diseño de la lata.

```
package AlgGenetico;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Random;

public class Lata {

	public static void main(String[] args) {
		InputStreamReader isr = new InputStreamReader(System.in);
		BufferedReader br = new BufferedReader(isr);

		double probabilidad_cruce = 0.8, probabilidad_mutacion = 0.01;

		try {
			System.out.print("Costo del material de la lata por cm2\t\t");
			double costo = Double.parseDouble(br.readLine());
			System.out.print("Volumen de la lata:\t\t\t\t");
			int volumen = Integer.parseInt(br.readLine());

			int[][] poblacion = new int[10][10];
			double[][] datos_pob = new double[11][6];

			pob_inicial(poblacion);
			calcular(volumen, costo, poblacion, datos_pob);
			imprimir(poblacion, datos_pob);
			torneo(volumen, poblacion, datos_pob);
			calcular(volumen, costo, poblacion, datos_pob);
			imprimir(poblacion, datos_pob);
			for(int d = 0; d < 100; d++){
				torneo(volumen, poblacion, datos_pob);
				cruce(poblacion, probabilidad_cruce);
				mutacion(poblacion, probabilidad_mutacion);
				calcular(volumen, costo, poblacion, datos_pob);
//				imprimir(poblacion, datos_pob);
			}
			imprimir(poblacion, datos_pob);

		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	private static void mutacion(int[][] poblacion, double probabilidad_mutacion) {
		Random aleatorio = new Random();
```

```
		int i = poblacion.length;
		int j = poblacion[0].length;

		for (int a = 0; a < i; a++){
			double mutacion = aleatorio.nextDouble();

			if (mutacion < probabilidad_mutacion){
				int gen = aleatorio.nextInt(j);

				poblacion[a][gen] = Math.abs(poblacion[a][gen] - 1);
			}
		}
	}

	private static void cruce(int[][] poblacion, double probabilidad_cruce) {
		Random	aleatorio = new Random();

		int i = poblacion.length;
		int j = poblacion[0].length;
		int individuo1, individuo2, puntocruce;
		int[][] reproducciones = new int[i][j];
		double cruce;

		for (int a = 0; a < i; a++){
			individuo1 = aleatorio.nextInt(i);
			individuo2 = aleatorio.nextInt(i);
			cruce = aleatorio.nextDouble();
			puntocruce = aleatorio.nextInt(i - 1);

			if (cruce < probabilidad_cruce){
				for (int gen = 0; gen < j; gen++){
					if (gen < puntocruce){
						reproducciones[a][gen] =
poblacion[individuo2][gen];
					} else{
						reproducciones[a][gen] =
poblacion[individuo1][gen];
					}
				}
			} else{
				for (int gen = 0; gen < j; gen++){
					reproducciones[a][gen] = poblacion[individuo1][gen];
				}
			}
		}
```

```
                for(int b = 0; b < i; b++){
                        for (int c = 0; c < j; c++){
                                poblacion[b][c] = reproducciones[b][c];
                        }
                }
        }

        private static void torneo(int volumen, int[][] poblacion, double[][] datos_pob) {
                Random      aleatorio = new Random();

                int i = poblacion.length;
                int g = poblacion[0].length;
                int individuo1, individuo2;
                int[][] area_reproduccion = new int[i][g];

                for (int p = 0; p < i; p++){
                        if (datos_pob[p][3] < volumen){
                                int costo_extra = aleatorio.nextInt(g / 2);

                                poblacion[p][costo_extra] =
Math.abs(poblacion[p][costo_extra] - 1);
                        }
                }

                for (int a = 0; a < i; a++){
                        individuo1 = aleatorio.nextInt(i);
                        individuo2 = aleatorio.nextInt(i);

                        if ((datos_pob[individuo1][3] >= volumen) &&
(datos_pob[individuo1][2] < datos_pob[individuo2][2])){
                                for (int genes = 0; genes < g; genes++){
                                        area_reproduccion[a][genes] =
poblacion[individuo1][genes];
                                }
                        }
                        if ((datos_pob[individuo2][3] >= volumen) &&
(datos_pob[individuo2][2] < datos_pob[individuo1][2])){
                                for (int genes = 0; genes < g; genes++){
                                        area_reproduccion[a][genes] =
poblacion[individuo2][genes];
                                }
                        }
                        else{
                                for (int genes = 0; genes < g; genes++){
                                        area_reproduccion[a][genes] = poblacion[a][genes];
                                }
                        }
```

```
            }

            for (int b = 0; b < i; b++){
                for (int c = 0; c < g; c++){
                    poblacion[b][c] = area_reproduccion[b][c];
                }
            }
        }

        private static void imprimir(int[][] poblacion, double[][] datos_pob) {
            int i = poblacion.length;
            int g = poblacion[0].length;
            int c = datos_pob[0].length;

            for (int a = 0; a < i; a++){
                System.out.print(" [");
                for (int b = 0; b < g; b++){
                    System.out.print(" " + poblacion[a][b]);
                }
                System.out.print(" ]\t");
                for (int k = 0; k < c; k++){
                    System.out.print("[ " + datos_pob[a][k] + " ]   ");
                }
                System.out.println();
            }
            System.out.println();
        }

        private static void calcular(int volumen, double costo, int[][] poblacion, double[][]
datos_pob) {
            int i = poblacion.length;
            int j = poblacion[0].length;
            double suma = 0, acumulador = 0;

            for (int a = 0; a < i; a++){
                int potencia = j / 2 - 1;
                int individuo = 0;
                double decodificacion= 0;

                for (int b = 0; b < j; b++){
                    decodificacion = decodificacion + poblacion[a][b] *
                                        Math.pow(2, potencia);
                    if (potencia == 0){
                        datos_pob[a][individuo] = decodificacion;
                        potencia = j / 2;
                        decodificacion = 0;
                        individuo++;
```

```
                        }
                        potencia--;
                    }

                    datos_pob[a][2] = costo * (((Math.PI * Math.pow(datos_pob[a][0], 2))
                            / 2) + Math.PI * datos_pob[a][0] * datos_pob[a][1]);
                    datos_pob[a][3] = (Math.PI * Math.pow(datos_pob[a][0], 2) *
datos_pob[a][1]) / 4;
                    suma = suma + datos_pob[a][2];
                }
                datos_pob[i][0] = suma;
                //Suma
                datos_pob[i][1] = suma / i;
        //Media

                suma = 0;
                for (int c = 0; c < i; c++){
                    datos_pob[c][4] = datos_pob[c][2] / datos_pob[i][0];
                    acumulador += datos_pob[c][4];
                    datos_pob[c][5] = acumulador;
                    suma += Math.pow(datos_pob[c][2] - datos_pob[i][1], 2);
                }
                datos_pob[i][2] = Math.sqrt(suma / (i - 1));
        //Desviación Estándar
        }

        private static void pob_inicial(int[][] poblacion) {
                Random                          gen = new Random();

                int i = poblacion.length;
                int j = poblacion[0].length;

                for (int x = 0; x < i; x++){
                    for (int y = 0; y < j; y++){
                        poblacion[x][y] = gen.nextInt(2);
                    }
                }
        }
}
```

3.6 Conclusiones y recomendaciones.

El poder de los Algoritmos Genéticos proviene del hecho de que se trata de una técnica robusta, y puede tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades.

Si bien no se garantiza que el Algoritmo Genético encuentre la solución óptima, del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de algoritmos de optimización combinatoria. En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al Algoritmo Genético, tanto en rapidez como en eficacia.

El gran campo de aplicación de los Algoritmos Genéticos se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en casos en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de las mismas hibridándolas con los Algoritmos Genéticos.

En el caso de los ejemplos de los algoritmos genéticos con los que se trabajaron en el presente reporte, se usaron funciones objetivo que bien pueden ser resueltas fácilmente por otras técnicas; la intención es mostrar como es el funcionamiento del código de programación. En la vida real, los algoritmos genéticos son usados en situaciones más complejas o de difícil resolución, donde es necesario computar gran cantidad de variables o no se pueden resolver por métodos comunes.

3.7 Referencias Bibliográficas y citas.

[1] Fu, K. S., González, R. C., Lee, C. S. G. “Robótica: Control, Detección, Visión e Inteligencia”. McGraw-Hill. 1988.

[2] Darwin, Charles. “On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life”. 1859.

[3] Mendel, Gregor. “Experiments in Plant Hibridization”. Brünn Natural History Society. 1865.

[4] Melero, Juan Julián. “Informática evolutiva”. 2005.

[5] García, Francisco Javier. “Evolución de la informática”. 2000.

[6] Holland, Jhon H. “Adaptation in Natural and Artificial Systems”. University of Michigan Press. 1975.

[7] Goldberg, David E. “Genetic Algorithms in Search, Optimization, and Machine Learning”. Addison – Wesley Publishing Company. 1989.

[8] Pajares, Gonzalo. “Visión por computador”. 2008.

[9] http://geneura.ugr.es/~jmerelo/ie/intro.htm

[10] http://flanagan.ugr.es/oep/node9.html