

ESCUELA SUPERIOR POLITECNICA DEL LITORAL

INSTITUTO DE CIENCIAS MATEMATICAS

TESIS DE GRADO

**Previa a la obtención del título de
INGENIERO EN ESTADÍSTICA INFORMÁTICA**

“Modelo poblacional con algoritmos genéticos”

**Presentada por:
Eduardo José Véliz Quintero**



Guayaquil – Ecuador

Año: 2001

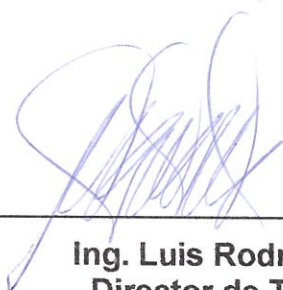
AGRADECIMIENTO

Agradezco a Dios por haberme dado la capacidad y constancia durante estos años de esfuerzo. A mis padres y hermanos por el apoyo permanente durante toda mi carrera universitaria, hasta hoy. Al Ing. Gaudencio Zurita Herrera por compartir conmigo y con mis compañeros sus conocimientos y experiencia. Al Ing. Luis Rodríguez Ojeda por haberme apoyado siempre y por haber sido mi guía en el desarrollo de esta tesis.

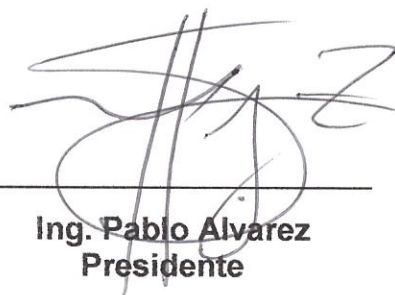
DEDICATORIA

Dedico esta tesis a mi familia y amigos más queridos.

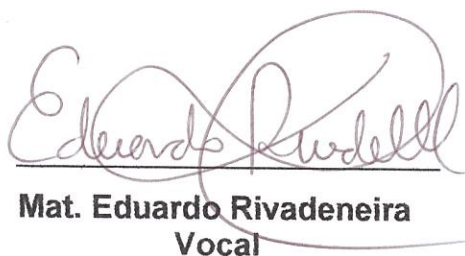
TRIBUNAL DE GRADUACION



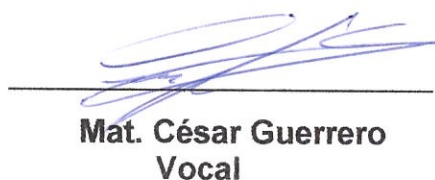
Ing. Luis Rodríguez
Director de Tesis



Ing. Pablo Alvarez
Presidente



Mat. Eduardo Rivadeneira
Vocal



Mat. César Guerrero
Vocal

DECLARACION EXPRESA

"La responsabilidad del contenido de esta tesis de grado, me corresponde exclusivamente; y su patrimonio intelectual a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL"



ING. EN ESTADISTICA INFORMATICA
Eduardo Véliz Q.

Eduardo José Véliz Quintero

RESUMEN

Para el desarrollo y elaboración de este trabajo, “MODELO POBLACIONAL CON ALGORITMOS GENÉTICOS”, he hecho una investigación de una rama de la inteligencia artificial, así como son los algoritmos genéticos. Lo que se desea es implantar un modelo de crecimiento poblacional mediante el uso de estos como una herramienta en la simulación.

En el primer capítulo presento en forma muy general los aspectos que envuelven los algoritmos genéticos, parto de la necesidad de optimizar mostrando las ventajas y desventajas de utilizar los algoritmos genéticos, así como su historia y posibles aplicaciones.

En el segundo capítulo he cubierto detalladamente todo lo que pude investigar sobre la teoría de los algoritmos genéticos, esto es, sus fundamentos matemáticos, tipos de algoritmos genéticos, representación del mismo, partes y operadores más importantes de los algoritmos genéticos. Con esto quiero dejar un trabajo sobre los algoritmos genéticos que pueda ser consultado y satisfaga realmente las necesidades. Además se presenta una pequeñísima comparación con otros métodos de programación evolutiva, tales como los algoritmos genéticos paralelos, también llamados algoritmos

meméticos, los sistemas de redes neuronales artificiales, y los autómatas celulares.

En el tercer capítulo hago un estudio de los principales modelos de crecimiento poblacional, que son aplicados tanto a humanos como a animales, luego propongo un modelo basado en el paradigma de los algoritmos genéticos.

En el cuarto capítulo llevo a cabo una simulación poblacional utilizando una aplicación que he desarrollado completamente en el lenguaje de cuarta generación, Visual C++ 6.0. Además implanto dos aplicaciones para estimar los parámetros de los modelos exponencial y logístico de un grupo de datos.

En el quinto y último capítulo, realizo una comparación entre mi modelo de crecimiento poblacional y los modelos convencionales, basándome en un subconjunto de datos de los que son generados por mi modelo, y realizar una predicción de los restantes.

Finalmente escribo las conclusiones y recomendaciones de mi estudio, en las que involucro toda la experiencia asimilada, con la finalidad de que sea más fácil la investigación acerca de este trabajo por parte de otras personas.

INDICE GENERAL

AGRADECIMIENTO	II
DEDICATORIA	III
TRIBUNAL DE GRADUACION	IV
DECLARACION EXPRESA	V
RESUMEN	VI
INDICE GENERAL	VIII
INDICE DE TABLAS	XIII
INDICE DE FIGURAS	XVI
INDICE DE CUADROS	XXI
INDICE DE ANEXOS	XXIII
INTRODUCCION	1
CAPITULO I	
1. Introducción: Los algoritmos genéticos	3
1.1 Introducción.	3
1.2 Funcionamiento de los algoritmos genéticos.	12
1.3 Ventajas y desventajas de los algoritmos genéticos.	13
1.4 Aplicaciones que puede realizarse con los algoritmos genéticos.	15
1.4.1 Ejemplos de aplicaciones realizadas.	16
1.4.2 Ejemplos de otras aplicaciones que pueden realizarse.	18
1.5 Historia.	21
1.5.1 Personajes.	22
1.5.2 La evolución.	28

1.5.3	La informática evolutiva.	32
1.6	Ubicación del problema.	40
1.7	Limitaciones y alcance del tema.	41
1.8	Objetivos: generales y específicos.	41

CAPITULO II

2.	Teoría de los algoritmos genéticos	43
2.1.	Introducción.	43
2.2.	Marco teórico.	44
2.2.1.	El teorema de esquemas.	44
2.2.1.1.	El orden de los esquemas.	44
2.2.1.2.	La longitud de los esquemas	45
2.2.2.	Paralelismo implícito.	50
2.2.3.	Teoremas de convergencia	51
2.3.	La estructura de un algoritmo genético	54
2.4.	Tipos de algoritmos genéticos.	55
2.4.1.	Algoritmos genéticos generacionales.	58
2.4.2.	Algoritmos genéticos de estado fijo.	58
2.4.3.	Algoritmos genéticos paralelos.	60
2.4.3.1.	Modelos de Islas.	62
2.4.3.2	Modelo celular.	62
2.5.	El algoritmo genético básico.	63
2.5.1.	La Población Inicial.	65
2.5.1.1.	Codificación Binaria.	65
2.5.1.2.	Codificación por permutaciones.	66
2.5.1.3.	Codificación por valores.	67
2.5.1.4.	Codificación mediante árboles.	67
2.5.2.	El genotipo	68
2.5.3.	La función de adaptación	68
2.5.4.	La selección de padres.	72

2.5.4.1.	Modelo de selección proporcional a la función objetivo.	73
2.5.4.2.	Modelo de selección proporcional al rango.	74
2.5.4.3.	Modelo de selección del valor esperado.	74
2.5.4.4.	Modelo de muestreo con remplazo del resto.	75
2.5.4.5.	Modelo del muestreo universal estocástico.	75
2.5.4.6.	Modelo de la rueda de la ruleta.	76
2.5.4.7.	Modelo del Torneo.	77
2.5.4.8.	Modelo de mantener el estado.	77
2.5.4.9.	Elitismo	78
2.6.	Crossover.	79
2.6.1.	Crossover de la codificación Binaria.	79
2.6.1.1.	Crossover en n puntos.	80
2.6.1.2.	Crossover Uniforme.	80
2.6.1.3.	Crossover matemático.	81
2.6.1.4.	Máscara de crossover	81
2.6.2.	Crossover de la codificación por permutaciones.	82
2.6.2.1.	Crossover en un solo punto.	82
2.6.3.	Crossover de la codificación por valores.	82
2.6.3.1.	Crossover en un punto.	83
2.6.3.2.	Crossover en n puntos.	83
2.6.3.3.	Crossover uniforme.	84
2.6.4.	Crossover de la codificación mediante árboles.	84
2.7.	Mutación	85
2.7.1.	Mutación en codificación binaria.	86
2.7.1.1.	Mutación en n puntos.	86
2.7.1.2.	Mutación uniforme.	86
2.7.2.	Mutación en codificación por permutaciones.	87
2.7.3.	Mutación en codificación por valores.	87

2.7.4.	Mutación en codificación por árboles.	88
2.8.	Otros operadores.	88
2.8.1.	Cromosomas de longitud variable.	88
2.8.2.	Operadores de nicho (ecológico).	90
2.8.3.	Operadores especiales.	91
2.9.	La población.	92
2.9.1.	Técnicas basadas en población.	92
2.9.2.	Tamaño de la población.	93
2.9.3.	La población inicial.	94
2.10.	Otras técnicas de programación evolutiva.	94
2.10.1.	Redes neuronales.	94
2.10.1.1.	La neurona	95
2.10.1.2.	La red neuronal	96
2.10.1.3.	Las redes neuronales artificiales	97
2.10.1.3.1.	Sistema experto.	97
2.10.1.3.2.	Interpretación de la neurona por computadora.	98
2.10.1.3.2.1.	La neurona artificial.	98
2.10.1.3.2.2.	Entrenamiento	101
2.10.1.4.	Método de transmisión de la información en el cerebro	102
2.10.1.4.1.	Compuertas lógicas.	103
2.10.1.4.1.1.	Funcionamiento de las sinapsis.	103
2.10.1.5.	Diferencias entre el cerebro y un ordenador	105
2.10.1.6.	Similitudes entre el cerebro y una computadora	107
2.10.1.7.	Un super ordenador llamado cerebro	107
2.10.1.8.	Aplicaciones de las redes neuronales	110
2.10.1.9.	Comparación entre los algoritmos genéticos y las redes neuronales.	111
2.11.	Autómatas Celulares.	111

2.11.1.	Comparación entre los algoritmos genéticos y los autómatas celulares	115
2.11.2.	Aplicaciones de los autómatas celulares.	124
2.12.	Observaciones	124

CAPITULO III

3.	Modelo poblacional: variables poblacionales	125
3.1.	Introducción.	125
3.2.	La población.	125
3.2.1.	Interacción entre las poblaciones.	126
3.2.2.	Los modelos poblacionales.	126
3.2.2.1.	Crecimiento demográfico.	132
3.2.3.	Superpoblación y recursos.	136
3.2.3.1.	El cero es buena nota.	138
3.2.4.	Modelos de crecimiento poblacional.	140
3.2.4.1.	Modelo de crecimiento exponencial.	144
3.2.4.2.	Modelo logístico de crecimiento.	149
3.2.4.3.	Población en crecimiento y población en retracción.	154
3.2.4.4.	. Características Comparativas de los modelos exponencial y logístico	155
3.2.4.5.	Modelo poblacional utilizando los algoritmos genéticos.	156

CAPITULO IV

4.	Simulación poblacional con algoritmos genéticos	169
4.1.	Introducción.	169
4.2.	Manual del Usuario.	171
4.3.	Manual Técnico.	195
4.3.1.	Diagramas a nivel de contexto.	195
4.3.2.	Diagramas de nivel 0.	198

4.3.3.	Descripción de las variables y procesos de los algoritmos genéticos.	203
4.3.3.1.	Descripción de las variables y procesos del algoritmo genético de simulación.	203
4.3.3.2.	Descripción de las variables y procesos del algoritmo genético de Optimización para los modelos dependientes del tiempo	212
4.3.3.3.	Descripción de las variables y procesos del algoritmo genético de Optimización para los modelos dependientes de de varias variables	218

CAPITULO V

5.	Análisis de las variables	224
5.1	Introducción	224
5.2	Análisis de los resultados de la aplicación: Algoritmos genéticos – modelo poblacional.	225
5.3	Predicción de los resultados por medio de los métodos convencionales.	255
5.4	Predicción de los resultados mediante algoritmos genéticos.	263
5.5	Selección del mejor modelo mediante algoritmos genéticos	266
5.6	Observaciones.	267

Conclusiones y recomendaciones

ANEXOS

BIBLIOGRAFIA

INDICE DE TABLAS

Tabla I	Tabla de características comparativas entre los modelos exponencial y logístico	155
Tabla II	Características del modelo poblacional con algoritmos genéticos	164
Tabla III	Tabla de la simulación con dos hijos, crossover en un punto, mutación en un punto, selección con el método de la rueda de la ruleta, sin elitismo, sin exigir mutación	227
Tabla IV	Tabla de la simulación con dos hijos, crossover uniforme, mutación en un punto, selección con el método de la rueda de la ruleta, sin elitismo, sin exigir mutación.	229
Tabla V	Tabla de la simulación con dos hijos, crossover en un punto, mutación uniforme, selección con el método de la rueda de la ruleta, sin elitismo, sin exigir mutación.	231
Tabla VI	Tabla de la simulación con dos hijos, crossover en un punto, mutación en un punto, selección con el método de la rueda de la ruleta, con elitismo, sin exigir mutación.	233
Tabla VII	Tabla de la simulación con dos hijos, crossover en un punto, mutación en un punto, selección con el método de la rueda de la ruleta, sin elitismo y exigiendo la mutación.	235
Tabla VIII	Tabla de la simulación con dos hijos, crossover en un punto, mutación en un punto, selección con el método del torneo, sin elitismo y sin exigir la mutación.	237

Tabla IX	Tabla de la simulación con cuatro hijos, crossover en un punto, mutación en un punto, selección con el método de la rueda de la ruleta, sin elitismo, sin exigir mutación.	239
Tabla X	Tabla de la simulación con cuatro hijos, crossover uniforme, mutación en un punto, selección con el método de la rueda de la ruleta, sin elitismo, sin exigir mutación.	241
Tabla XI	Tabla de la simulación con cuatro hijos, crossover en un punto, mutación en un punto, selección con el método de la rueda de la ruleta, con elitismo, sin exigir mutación.	243
Tabla XII	Tabla de la simulación con cuatro hijos, crossover en un punto, mutación en un punto, selección con el método de la rueda de la ruleta, sin elitismo, con exigencia de la mutación.	245
Tabla XIII	Tabla de la simulación con cuatro hijos, crossover en un punto, mutación en un punto, selección con el método del torneo, sin elitismo, sin exigir de la mutación.	247
Tabla XIV	Tabla de la simulación con el límite de vida disminuido a 50 años.	249
Tabla XV	Tabla de la simulación con el intervalo de fecundidad disminuido desde los 25 a los 30 años únicamente.	251
Tabla XVI	Tabla de la simulación limitando el crecimiento de la población	253
Tabla XVII	Tabla de las iteraciones 1 a 17 de la simulación para comparar	256

Tabla XVIII	Tabla de las iteraciones 18 a 28 de la simulación para comparar	257
Tabla XIX	Tabla de las iteraciones 29 a 48 de la simulación para comparar	257
Tabla XX	Valores estimados por el modelo exponencial	262
Tabla XXI	Valores estimados por el modelo lineal	262
Tabla XXII	Valores enteros estimados por el modelo lineal	262
Tabla XXIII	Primera estimación de B_0 y B_1 para el modelo exponencial	263
Tabla XXIV	Segunda estimación de B_0 y B_1 para el modelo exponencial	264
Tabla XXV	Tercera estimación de B_0 y B_1 para el modelo exponencial	265
Tabla XXVI	Datos para realizar la ejecución de la selección del mejor modelo de varias variables.	266
Tabla XXVII	Resultados de la ejecución del algoritmo genético para seleccionar el modelo.	267

INDICE DE FIGURAS

Figura 1.1	Carl Linnaes	22
Figura 1.2	Thomas Robert Malthus	23
Figura 1.3	Eramus Darwin	24
Figura 1.4	Charles Darwin	25
Figura 1.5	Alfred Russel Wallace	26
Figura 1.6	John H. Holland	27
Figura 1.7	David Golberg	28
Figura 2.1	Diagrama de flujo del algoritmo genético básico	64
Figura 2.2	Ejemplo de un cromosoma de la codificación por árboles	67
Figura 2.3	Gráfico de la selección de la rueda de la ruleta	76
Figura 2.4	Ejemplo de crossover en la representación mediante arboles	84
Figura 2.5	Ejemplo de la mutación en la representación por árboles	88
Figura 2.6	Función de transferencia de las redes neuronales de tipo umbral lógico	100
Figura 2.7	Ejemplo de una función de transferencia de las redes neuronales de tipo S	100
Figura 2.8	Ejemplo de una función de transferencia de las redes neuronales de tipo limitación dura	100
Figura 2.9	Tipos de funciones en cuanto a linealidad y continuidad	117
Figura 3.1	Gráfico de las tasas de natalidad, mortalidad y crecimiento	133
Figura 3.2	Gráfico de las tasas de fertilidad de 1998	135
Figura 3.3	Población Separada en hombres y mujeres con n personas.	164
Figura 3.4	Estructura de tipo población	165

Figura 3.5	Estructura de tipo persona	165
Figura 3.6	Estructura de tipo datos de persona	166
Figura 3.7	Estructura de Tipo Genético	166
Figura 4.1	Pantalla inicial de la aplicación AgMp de Algoritmos Genéticos – Modelo Poblacional	172
Figura 4.2	Pantalla Principal del Simulador del AgMp, Algoritmos Genéticos – Modelo Poblacional	174
Figura 4.3	Línea de progreso de la ejecución del Algoritmos Genéticos – Modelo Poblacional	175
Figura 4.4	Pantalla de los resultados del Simulador de los Algoritmos Genéticos – Modelo Poblacional	176
Figura 4.5	Pantalla con el Gráfico del tamaño de la población a través del tiempo	177
Figura 4.6	Gráfico de las aptitudes de los individuos de la población a través del tiempo	178
Figura 4.7	Pantalla del Gráfico de Crecimiento y decrecimiento porcentual a través del tiempo del Simulador del Algoritmos Genéticos – Modelo Poblacional	179
Figura 4.8	Pantalla del Gráfico de las Edades promedio de la población a través del tiempo del Simulador del Algoritmos Genéticos – Modelo Poblacional	181
Figura 4.9	Pantalla del Gráfico de la población por Edades y por sexo en el último año de simulación del Algoritmos Genéticos – Modelo Poblacional	182
Figura 4.10	Pantalla Principal del Modelo dependiente del tiempo del Algoritmos Genéticos – Modelo Poblacional	183
Figura 4.11	Pantalla de resultados para el modelo dependiente del tiempo mediante algoritmos genéticos	184
Figura 4.12	Pantalla del Gráfico de los estimadores del modelo	185

	dependiente del tiempo.	
Figura 4.13	Pantalla del Gráfico de los errores del modelo dependiente del tiempo.	186
Figura 4.14	Pantalla de los intervalos de confianza del modelo dependiente del tiempo.	187
Figura 4.15	Pantalla Principal del Modelo dependiente de varias variables del Algoritmos Genéticos – Modelo Poblacional	188
Figura 4.16	Pantalla de resultados para los modelos dependientes de varias variables y mediante algoritmos genéticos	189
Figura 4.17	Pantalla del Gráfico de la tendencia del modelo de crecimiento exponencial o logístico dependientes de varias variables.	190
Figura 4.18	Pantalla del Gráfico de los errores de los modelos de crecimiento exponencial o logístico pero dependientes varias variables.	191
Figura 4.19	Pantalla de la apertura de un archivo	192
Figura 4.20	Mensaje de la apertura de un archivo	192
Figura 4.21	Pantalla de la configuración avanzada para el crossover	193
Figura 4.22	Pantalla de la configuración avanzada para la mutación	194
Figura 4.23	Diagrama de contexto del algoritmo genético de simulación	195
Figura 4.24	Diagrama de contexto del algoritmo genético del modelo dependiente del tiempo	196
Figura 4.25	Diagrama de contexto del algoritmo genético del modelo dependiente de varias variables	197
Figura 4.26	Diagrama cero del algoritmo genético de simulación	198
Figura 4.27	Diagrama cero del algoritmo genético del modelo dependiente del tiempo.	199

Figura 4.28	Diagrama de nivel 1 del proceso de cálculo del algoritmo genético del modelo dependiente del tiempo para los beta.	200
Figura 4.29	Diagrama cero del algoritmo genético del modelo dependiente de varias variables.	201
Figura 4.30	Diagrama de nivel 1 del proceso de cálculo del algoritmo genético del modelo de selección.	202
Figura 5.1	Tamaño poblacional de la primera simulación con dos hijos	228
Figura 5.2	Tamaño poblacional de la segunda simulación con dos hijos	230
Figura 5.3	Tamaño poblacional de la tercera simulación con dos hijos	232
Figura 5.4	Tamaño poblacional de la cuarta simulación con dos hijos	234
Figura 5.5	Tamaño poblacional de la quinta simulación con dos hijos	236
Figura 5.6	Tamaño poblacional de la sexta simulación con dos hijos	238
Figura 5.7	Tamaño poblacional de la primera simulación con cuatro hijos	240
Figura 5.8	Tamaño poblacional de la segunda simulación con cuatro hijos	242
Figura 5.9	Tamaño poblacional de la tercera simulación con cuatro hijos	244
Figura 5.10	Tamaño poblacional de la cuarta simulación con cuatro hijos	246
Figura 5.11	Tamaño poblacional de la quinta simulación con cuatro hijos	248

Figura 5.12	Tamaño poblacional de la sexta simulación con cuatro hijos	250
Figura 5.13	Tamaño poblacional de la séptima simulación con cuatro hijos	252
Figura 5.14	Tamaño poblacional de la octava simulación con cuatro hijos	254
Figura 5.15	Línea a través del tiempo de los datos simulados	258
Figura 5.16	Comparación con las tendencias lineal y exponencial	260
Figura 5.17	Residuos de los datos ajustados a los modelos lineal y exponencial	261

INDICE DE CUADROS

Cuadro 2.1	El algoritmo genético básico	63
Cuadro 2.2	Ejemplo de dos cromosomas en codificación binaria	66
Cuadro 2.3	Dos cromosomas en la representación por permutaciones	66
Cuadro 2.4	Tres cromosomas en codificación por valores	67
Cuadro 2.5	Algoritmo para realizar el método de selección de la rueda de la ruleta	76
Cuadro 2.6	Crossover en un punto de la representación binaria	80
Cuadro 2.7	Crossover uniforme en la representación binaria	80
Cuadro 2.8	Crossover matemático en la representación binaria	81
Cuadro 2.9	Crossover en un solo punto de la representación por permutaciones	82
Cuadro 2.10	Crossover en un solo punto de la representación por valores	83
Cuadro 2.11	Crossover en n puntos de la representación por valores	83
Cuadro 2.12	Crossover uniforme en la representación por valores	84
Cuadro 2.13	Mutación en n puntos de la representación binaria	86
Cuadro 2.14	Mutación uniforme de la representación binaria	87
Cuadro 2.15	Mutación en la representación por permutaciones	87
Cuadro 2.16	Mutación en la representación por valores	87
Cuadro 3.1	Algoritmo genético aplicado a los modelos poblacionales	168
Cuadro 4.1	Estructura de tipo genético	204
Cuadro 4.2	Estructura de tipo datos de la persona	205

Cuadro 4.3	Estructura de tipo persona	206
Cuadro 4.4	Estructura de tipo población	207
Cuadro 4.5	Estructura de tipo selección	207
Cuadro 4.6	Estructura de tipo persona (tiempo)	214
Cuadro 4.7	Estructura de tipo población (tiempo)	215
Cuadro 4.8	Estructura de tipo persona (Selección)	219
Cuadro 4.9	Estructura de tipo población (Selección)	220

INDICE DE ANEXOS

- Anexo 1:** Código fuente del Algoritmo Genético Básico.
- Anexo 2:** Ejecución del Algoritmo Genético Básico.
- Anexo 3:** Código fuente del Problema del vendedor viajero (TSP).
- Anexo 4:** Ejecución del Problema del vendedor viajero (TSP).
- Anexo 5:** Código fuente de una Simulación en tiempo Real.
- Anexo 6:** Ejecución de una Simulación en Tiempo Real.
- Anexo 7:** Código fuente del Algoritmo Genético – Modelo Poblacional .

INTRODUCCION

En la naturaleza, los individuos compiten entre sí por recursos, tales como comida, agua y refugio. Adicionalmente, entre los animales de una misma especie, aquellos que no obtienen acierto tienden a tener un número reducido de descendientes, teniendo por tanto, menor probabilidad de que sus genes sean propagados a lo largo de sucesivas generaciones.

Los algoritmos genéticos es una técnica de búsqueda basada en la teoría de la evolución de Darwin, que ha cobrado tremenda popularidad en todo el mundo durante los últimos años. Aquí presento los conceptos básicos que se requieren para comprenderla, así como unos ejemplos que permitan a los lectores comprender cómo aplicarla al problema de su elección.

Un aspecto por demás importante de ellos es su aplicación como técnica de optimización que se basa en el azar, pero que aprovecha criterios que la naturaleza ha desarrollado, tales como la selección de los cromosomas más aptos, el cruce de genes en los cromosomas y la mutación. Por esto, no es de extrañarse que en algoritmos genéticos se utilicen términos tomados de la genética natural.

Un algoritmo genético puede converger en una búsqueda de azar, pero su utilización asegura que ningún punto del espacio de búsqueda tiene probabilidad cero de ser examinado.

En el desarrollo de esta tesis utilizo la técnica de los algoritmos genéticos para implantar un modelo de crecimiento poblacional algorítmico de gran robustez que permite la simulación, así como la optimización de sus parámetros.

Para mayor comprensión de la presente tesis de parte de cualquier lector, presento una explicación modesta de los modelos de crecimiento poblacional, aunque es también bastante buena y suficiente para el aprendizaje de los mismos, y la posterior aplicación de los algoritmos genéticos a los modelos poblacionales.

CAPITULO I

1. Introducción: Los algoritmos genéticos

1.1. Introducción

En este primer capítulo se relata la utilidad de los algoritmos genéticos, así como su uso en problemas de optimización y de simulación. Además se explica un poco de la historia de los algoritmos genéticos, y se nombra diversas aplicaciones de los mismos, de las cuales he podido desarrollar unos cuantos ejemplos para la comprensión de los mismos.

Se conoce que muchos problemas pueden ser resueltos de una forma computacional determinística, mientras que otros no tienen un método de resolución exacta y utilizan métodos numéricos o técnicas computacionales; pero todos estos métodos de resolución son complejos en su implantación y uso. Para la solución de éstos

puede utilizarse métodos heurísticos y metaheurísticos como los algoritmos genéticos.

Los algoritmos genéticos forman parte de la inteligencia artificial de los sistemas inspirados en la naturaleza y la evolución, y son métodos que simulan los procesos naturales y los aplican a la solución de problemas reales de búsqueda, optimización, diseño y simulación, aplicando la idea darwiniana de la selección natural de acuerdo a la aptitud y la disposición, y la combinan con otros operadores genéticos para producir métodos de gran robustez y de aplicación óptima.

Los algoritmos genéticos es una técnica de búsqueda basada en la teoría de la evolución de Darwin, que ha cobrado tremenda popularidad en todo el mundo durante los últimos años. Se presentarán aquí los conceptos básicos que se requieren para abordarla, así como unos sencillos ejemplos que permitan a los lectores comprender cómo aplicarla al problema de su elección.

En los últimos años, la comunidad científica internacional ha mostrado un creciente interés en una nueva técnica de búsqueda basada en la teoría de la evolución y que se conoce como el algoritmo genético. Esta técnica se basa en los mecanismos de selección que utiliza la naturaleza, de acuerdo a los cuales los

individuos más aptos de una población son los que sobreviven, al adaptarse más fácilmente a los cambios que se producen en su entorno. Hoy en día se sabe que estos cambios se efectúan en los genes de un individuo que es la unidad básica de codificación de cada uno de los atributos de un ser vivo, y que sus atributos más deseables, es decir los que le permiten adaptarse mejor a su entorno, se transmiten a sus descendientes cuando éste se reproduce sexualmente.

Una definición bastante completa de un algoritmo genético es la propuesta por John Koza: "Es un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto, y tras haberse presentado de forma natural una serie de operaciones genéticas de entre las que destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija que se ajusta al modelo de las cadenas de cromosomas, y se les asocia con una cierta función matemática que refleja su aptitud."

Un aspecto por demás importante de ellos es su aplicación como técnica de optimización que se basa en el azar, pero que aprovecha criterios que la naturaleza ha desarrollado, tales como la selección de los cromosomas más aptos, el cruce de genes en los cromosomas y la mutación. Por esto, no es de extrañarse que en algoritmos genéticos se utilicen términos tomados de la genética natural.

Estos algoritmos están basados en los procesos genéticos de los organismos biológicos, codificando una posible solución a un problema de cromosomas compuesto por cadena de bits y caracteres. Estos cromosomas representan individuos que son llevados a lo largo de varias generaciones, en forma similar a los problemas naturales, evolucionando de acuerdo con los principios de selección natural y de supervivencia de los más aptos, descritos por primera vez por Charles Darwin en su libro "El Origen de las Especies". Simulando estos procesos, los algoritmos genéticos son capaces de evolucionar soluciones de problemas del mundo real.

En la naturaleza, los individuos compiten entre sí por recursos, tales como comida, agua y refugio. Adicionalmente, entre los animales de una misma especie, aquellos que no obtienen acierto

tienden a tener un número reducido de descendientes, teniendo por tanto, menor probabilidad de que sus genes sean propagados a lo largo de sucesivas generaciones. La combinación entre los genes de los individuos que perduran en una especie, puede producir un nuevo individuo mucho mejor adaptado a las características de su medio ambiente.

Los algoritmos genéticos utilizan una analogía directa de este fenómeno de evolución en la naturaleza, donde cada individuo representa una posible solución para un problema dado. A cada individuo se atribuye una puntuación de adaptación, dependiendo de la respuesta dada al problema por este individuo. A los más adaptados se les da la oportunidad de reproducirse mediante cruces con otros individuos de la población, produciendo descendientes con características de ambas las partes.

Si un algoritmo genético es desarrollado correctamente, la población (conjunto de posibles respuestas) convergerá a una buena solución para el problema propuesto, tal vez hasta óptima. Los procesos que más contribuyen para la evolución son el cruce de genes en cromosomas y la adaptación basada en una selección y de reproducción. La mutación también tiene un papel significativo, en tanto, su importancia continúa siendo un asunto en

que los investigadores aún no entran en consenso, ya que al igual que en la naturaleza, debe darse con una posibilidad muy pequeña.

Un algoritmo genético puede converger en una búsqueda de azar, pero su utilización asegura que ningún punto del espacio de búsqueda tiene probabilidad cero de ser examinado. Toda tarea de búsqueda y optimización posee varios componentes, entre ellos se encuentran:

- El espacio de búsqueda, donde son consideradas todas las posibilidades de solución de un determinado problema, y
- La función de validación (o función de coste), es una manera de validar los miembros del espacio de búsqueda.

Como técnicas de búsqueda y optimización tradicionales comienzan con un único candidato que, iterativamente, es manipulado utilizando algunas técnicas heurísticas (estáticas) directamente asociadas al problema a ser solucionado y por otro lado, las técnicas de computación y evolución operan sobre una población de candidatos en paralelo. Así, ellos pueden satisfacer la búsqueda en diferentes áreas del espacio de solución, alocando un número de miembros apropiado para la búsqueda en varias regiones. Los algoritmos genéticos difieren de los métodos

tradicionales de búsqueda y optimización, principalmente en cuatro aspectos:

1. Los algoritmos genéticos trabajan con una codificación del conjunto de parámetros y no con los propios parámetros;
2. Los algoritmos genéticos trabajan con una población de varios puntos y no con un único punto;
3. Los algoritmos genéticos utilizan la misma función y no derivadas u otro conocimiento auxiliar para adaptarse al problema;
4. Los algoritmos genéticos utilizan reglas de transición probabilísticas y no determinísticas.

Los Algoritmos Genéticos son muy eficientes para búsqueda de soluciones óptimas, o aproximadamente óptimas (buenas soluciones), en una gran variedad de problemas, porque no imponen muchas de las limitaciones encontradas en los métodos de búsqueda tradicionales. Los investigadores se refieren a los algoritmos genéticos o a un algoritmo genético y no al algoritmo genético, porque los algoritmos genéticos son una clase de procedimientos con muchos pasos separados, y cada uno de estos pasos posee muchas variaciones posibles.

Antes de continuar ahondando en la técnica de los Algoritmos Genéticos he pensado que sería interesante dejarla situada dentro de un marco más amplio. Me refiero con esto a la rama de la Inteligencia Artificial que se ha denominado Computación Evolutiva.

El término Computación Evolutiva se refiere al estudio de los fundamentos y aplicaciones de ciertas técnicas heurísticas de búsqueda basadas en los principios naturales de la evolución. Una gran variedad de algoritmos evolutivos ha sido propuesta; pero principalmente pueden clasificarse en: Algoritmos Genéticos, Programación Evolutiva, Estrategias Evolutivas, Sistemas Clasificadores y Programación Genética.

Esta clasificación se basa sobre todo en detalles de desarrollo histórico más que en el hecho de un funcionamiento realmente diferente, de hecho las bases biológicas en las que se apoyan son esencialmente las mismas. Las diferencias entre ellos se centra en los operadores que se usan en cada caso y en general en la forma de implementar la selección, reproducción y sustitución de individuos en una población.

Aunque los detalles de la evolución no han sido completamente comprendidos, incluso hoy en día, existen algunos puntos en los que se fundamentan:

- La evolución es un proceso que opera a nivel de cromosomas, y no a nivel de individuos. Cada individuo es codificado como un conjunto de cromosomas.
- La selección natural es el mecanismo mediante el cual los individuos mejor adaptados son los que tienen mayores posibilidades de reproducirse.
- El proceso evolutivo tiene lugar en la etapa de la reproducción. Es en esta etapa donde se producen la mutación, que es la causante de que los cromosomas de los hijos puedan ser diferentes a los de los padres, y el cruce, que combina los cromosomas de los padres para que los hijos tengan cromosomas diferentes.

Los Algoritmos Genéticos resuelven los problemas generando poblaciones sucesivas a las que se aplican los operadores de mutación y cruce. Cada individuo representa una solución al problema, y se trata de encontrar al individuo que represente a la mejor solución.

La programación genética funciona igual que la técnica anterior pero se basa en el estudio de problemas cuya solución es un programa. De manera que los individuos de la población son programas que se acercan más o menos a realizar una tarea que es la solución.

La Programación Evolutiva es otro enfoque de los algoritmos genéticos, en este caso el estudio se centra en conseguir operadores genéticos que imiten lo mejor posible a la naturaleza, en cada caso, más que en la relación de los padres con su descendencia. En este caso no se utiliza el operador de cruce, tomando la máxima importancia el operador de mutación.

1.2. Funcionamiento de los algoritmos genéticos

El funcionamiento de un algoritmo genético se basa en la creación de un conjunto de posibles soluciones (población), representadas cada una de ellas como una cadena de símbolos que pueden ser caracteres, valores, o direcciones de memoria. Luego se califica a cada solución de acuerdo con una función de aptitud definida para el problema en cuestión. Con base en tal calificación se seleccionan elementos de ese conjunto para que se les apliquen los operadores genéticos. Dichos operadores tienen como objetivo crear nuevos individuos a partir de los anteriores,

conservando de esta forma las características que hicieron buenos a estos últimos.

Cada representación de las posibles soluciones depende del tipo de problema sobre el que se aplicará un algoritmo genético, es decir que para obtener un buen algoritmo genético debe implantarse una correcta representación, y luego una buena función de aptitud. Sólo de este modo, tendremos un algoritmo genético que convergerá a la solución deseada.

1.3. Ventajas y desventajas de los algoritmos genéticos

El hecho tener una población de posibles soluciones hace que el algoritmo genético esté explorando siempre varias soluciones en paralelo, y además se analizará cada una de las posibles soluciones, ya que cada punto tendrá igual probabilidad de ser seleccionado inicialmente. Es de aquí de donde aparece la dificultad de que un algoritmo genético se quede atrapado en un óptimo local. Esta es posiblemente una de las mayores ventajas de los algoritmos genéticos cuando se les ve como una alternativa en la optimización. Esta característica hace natural pensar en implementar algoritmos genéticos en sistemas de cómputo paralelo. Además, se debe considerar que la implantación de los algoritmos genéticos en algún lenguaje de programación es

relativamente simple, y aunque antes se pensaba que sólo se los podía implantar en lenguajes que utilicen el concepto de predicados; pero actualmente se han desarrollado muchos algoritmos genéticos en lenguajes hasta de cuarta generación, dándoles de esta manera un ambiente más apropiado a la actualidad.

Por otro lado, entre las desventajas de los algoritmos genéticos tenemos el hecho de que hay que diseñar funciones para representar las soluciones mediante cadenas y para calificar a los individuos de la población. Como eso depende del problema, pueden darse casos en que no sea tan fácil encontrarlas, o de aplicarlas en el problema. Además, los algoritmos genéticos son una herramienta de propósito general que normalmente son superados cuando existen algoritmos especializados para un problema particular.

En problemas donde obtener un resultado exacto es importante, el algoritmo genético puede servir simplemente como base para que luego otro algoritmo encuentre el óptimo, con una buena aproximación. En este caso el algoritmo genético lo que hace es encontrar un punto muy cercano a la solución, lo que posiblemente

a un algoritmo tradicional no le hubiera sido posible si la función analizada fuera muy compleja.

Los Sistemas de adaptación tratan de resolver problemas acumulando conocimiento sobre el mismo y utilizando estas informaciones para generar soluciones aceptables. Estos problemas, típicamente, se encuentran en las áreas de configuración de sistemas complejos, colocación de tarifas, selección de rutas, máximos, mínimos, búsquedas y otros problemas de optimización y aprendizaje de máquinas.

1.4. Aplicaciones que pueden realizarse con algoritmos genéticos.

La aplicación más común de los algoritmos genéticos ha sido la solución de problemas de optimización, en donde han mostrado ser muy eficientes y confiables. Sin embargo, no todos los problemas pudieran ser apropiados para la técnica, y se recomienda en general tomar en cuenta las siguientes características del mismo antes de intentar usarla:

- Su espacio de búsqueda, es decir sus posibles soluciones, debe estar delimitado dentro de un cierto rango.
- Debe poderse definir una función de aptitud que nos indique qué tan buena o mala es una cierta respuesta.

- Las soluciones deben codificarse de una forma que resulte relativamente fácil de implementar en la computadora.

El primer punto es muy importante, y lo más recomendable es intentar resolver problemas que tengan espacios de búsqueda discretos, aunque éstos sean muy grandes; sin embargo, también podrá intentarse usar la técnica con espacios de búsqueda continuos, pero preferentemente cuando exista un rango de soluciones relativamente pequeño para cubrir una mayor parte del mismo.

1.4.1. Ejemplos de aplicaciones realizadas:

Los siguientes son ejemplos de aplicaciones que he realizado mientras comprendía el funcionamiento de los algoritmos genéticos, y planteaba como utilizarlos en mi aplicación dirigida a los modelos de crecimiento poblacional.

- **El problema del vendedor viajero (TSP)**

Es una pequeña aplicación realizada en Visual C++ 6.0 que optimiza rutas. Esta aplicación lee un archivo de texto ubicado en el mismo directorio que la aplicación se encuentra. En esta aplicación, se utilizó una representación de los cromosomas por medio de permutaciones. Además debe configurarse como

parámetros del algoritmo, el tamaño de la población, el tamaño del torneo, el número de iteraciones y la semilla aleatoria. Estos son explicados en detalle en el capítulo II, para una mejor comprensión de los algoritmos genéticos.

- **Punto máximo en una función normal estándar**

Es una pequeña aplicación que me permitió comprender el algoritmo genético básico representando los cromosomas mediante cadenas binarias. Está desarrollado completamente en Visual C++ 6.0 en el modo ventana de diálogo con MFC, y pide el ingreso del número de iteraciones, tamaño de la población, intervalo de acción, probabilidad de crossover, probabilidad de mutación, y probabilidad de convergencia.

- **Algoritmos genéticos – modelo poblacional**

Es la principal aplicación que realicé, con la que determino el crecimiento de una población heterogénea, basada en el número de hijos, edad límite de vida, intervalo de edades de reproducción, y la configuración del algoritmo genérico que presenta algunas opciones que luego son explicadas detalladamente.

Este es ámbito al que nos vamos a dedicar en este proyecto, en este caso se aprovecha la selección natural, el cruce y la mutación para simular a través de generaciones, y observar las medidas de

desempeño de los índices poblacionales que nos interesan, tales como tasa de natalidad, tasa de mortalidad y las tasas de migraciones, así como la edad promedio de la población y la distribución por sexos.

1.4.2. Ejemplos de otras aplicaciones que pueden realizarse:

- **Planificación de multitarea**

Una aplicación de los algoritmos genéticos en un problema de asociación óptima de procesos y procesadores. Un objetivo es disminuir el costo que se deriva de la comunicación entre procesos en un ordenador paralelo de memoria distribuida. La Planificación de multitarea también puede ser relacionada con problemas de robótica.

- **Biología molecular y física o química**

Existe una hipótesis de trabajo que sustentaría el uso de técnicas basadas en poblaciones debido a una estructura de función objetivo. En problemas relacionados con estructura molecular existen experimentos que dan indicios claros sobre la validez de esa hipótesis. Así que es fácil observar que son cada vez más numerosas las aplicaciones de algoritmos genéticos en este campo.

- **Ingeniería en construcciones**

Los Algoritmos genéticos se han ganado la aceptación en un gran número de problemas de ingeniería, estos problemas están basados en la optimización de recursos. Una aplicación es la optimización discreta de estructuras.

- **Búsqueda en bases de datos**

Así como existen los métodos de búsqueda convencionales en bases de datos, los algoritmos genéticos pueden facilitar dichas búsquedas, dando resultados muy eficientes.

- **Geofísica**

Para un problema llamado de Inversión de la forma de la ola sísmica, generalmente se utiliza la estadística Bayesiana, a dicho problema se asocia una información que es apriori sobre un modelo. Surgen así funciones que requieren métodos de optimización global, donde es de mucho provecho el uso de los algoritmos genéticos.

- **Compresión de Datos**

La compresión de datos en general, es la compresión de imágenes sólidas en particular. Esta aplicación consiste en encontrar un método que utiliza los Algoritmos genéticos para encontrar un sistema de funciones locales iteradas (LIFS) para la

codificación de imágenes. Produciendo como resultado final una imagen con calidad similar a la utilización de un método convencional de compresión fractal, con un tiempo 30% menor.

- **Optimización de rutas**

Cuando queremos optimizar rutas, existen muchas técnicas de programación que nos llevan a una respuesta adecuada. Podemos aplicar también los algoritmos genéticos para hallar una ruta óptima.

Además existen aplicaciones de los algoritmos genéticos muy generales, que pueden ser adaptadas a cualquier necesidad, siempre que se tenga cuidado al considerar la correcta representación de sus elementos.

Entre estos algoritmos encontramos la aplicación GaUI 1.0, la cual está desarrollada completamente en un lenguaje de cuarto nivel, y presenta muchas etapas de los algoritmos genéticos para la resolución de diferentes problemas comunes.

1.5. Historia

Para comprender mejor la historia de los algoritmos genéticos, se ha dividido en dos partes muy claras como son la evolución y la informática evolutiva, en la primera se relatan los hechos evolutivos más notables, mientras en la segunda se comentan los hechos más sobresalientes de la informática evolutiva.

Se desea estudiar qué es lo que inspira dichos algoritmos, la naturaleza, ese fenómeno natural denominado evolución, y si de veras optimiza o no. Además es importante conocer un poco más sobre los personajes que destacan en esta reseña de la historia de los algoritmos genéticos.

1.5.1. Personajes



Carl Linnaeus

Figura 1.1

Carl Linnaeus (1707-1778), Conocido también como Carolus Linnaeus, es conocido como el padre de la Taxonomía, su sistema de nombramiento, discriminación y clasificación de organismos es usado en la actualidad con algunos pequeños cambios. Sus ideas de clasificación han influenciado a algunas generaciones de biólogos, incluso aquellos que se opusieron a las raíces filosóficas y teológicas de su trabajo.



Thomas Robert Malthus

Figura 1.2

Thomas Robert Malthus(1766 - 1834), Economista británico, discípulo de Adam Smith. Expuso sus teorías en la obra "Primer Ensayo sobre la población" (1798).

Se señala que de acuerdo a Malthus la población suele aumentar en una proporción geométrica y la producción de alimentos sólo puede aumentar en una proporción aritmética.

En este cuadro las diversas medidas de control de natalidad se convierten en un factor clave en la lucha por el desarrollo, aun cuando no se llega a asegurar que controlado el crecimiento de la población el progreso será realmente posible.

La aparición de Malthus en este trabajo es importante, ya que es un personaje que está involucrado tanto en la historia de los algoritmos genéticos como en la de los modelos poblacionales.



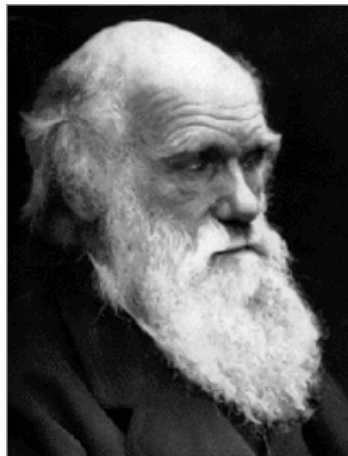
Erasmus Darwin

Figura 1.3

Erasmus Darwin (1731 – 1802), Era el abuelo de Charles Darwin, fue uno de los principales intelectuales de decimotercero siglo Inglaterra, un hombre con una serie notable de intereses y persecuciones.

Como un naturalista, él formuló uno de las primeras teorías formales en evolución en Zoonomía, o, Las Leyes de Vida Orgánica (1794-1796). Él también presentó sus ideas evolutivas

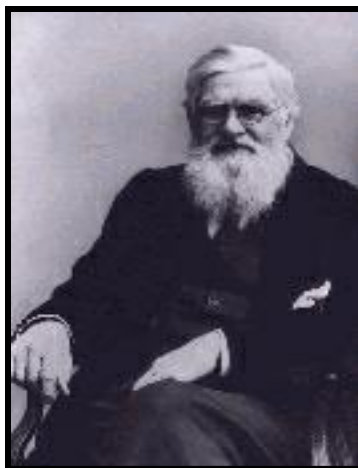
en verso, en particular en el poema póstumamente publicado El Templo de Naturaleza.



Charles Darwin

Figura 1.4

Charles Darwin(1809 –1882), nació el 12 de febrero de 1809 en Shrewsbury, Inglaterra. Él era el británico naturalista que tomó fama por sus teorías de evolución y la selección natural. Como varios científicos antes que él, Darwin creyó toda la vida en tierra evolucionada (desarrollado gradualmente) sobre de millones de años de unos antepasados comunes.



Alfred Russel Wallace

Figura 1.5

Alfred Russel Wallace (1823 - 1913) es uno de los padres olvidados de ciencia moderna. Él nació en el pueblo de Usk en Monmouthshire, Inglaterra. Su padre se murió cuando Alfred era joven.

Estuvo en Singapur desde 1853 y durante los próximos ocho años que Alfred Russel Wallace hizo el gran viaje que llevó a su formulación de la teoría de Selección Natural. Los nombres de Charles Robert Darwin (1809-1882) y Alfred Russel Wallace (1823-1913) tienen gran sentido con el concepto moderno de evolución y la teoría de selección natural.

En 1900, la teoría moderna de evolución combina la genética y las ideas de Darwin y Wallace sobre la selección natural, creando el

principio básico de Genética Poblacional: la variabilidad entre individuos en una población de organismos que se reproducen sexualmente es producida por la mutación y por la recombinación genética.



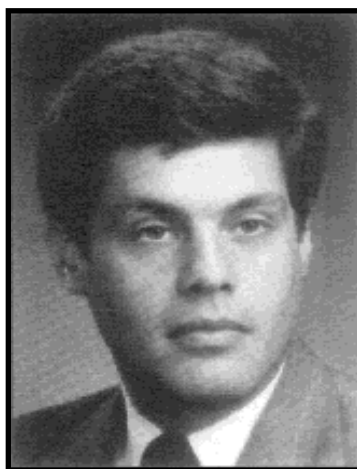
John H. Holland

Figura 1.6

John Holland, Se le conoce como el padre de los algoritmos genéticos. John Holland desde pequeño, se preguntaba cómo logra la naturaleza, crear seres cada vez más perfectos.

Fue a principios de los 60, en la Universidad de Michigan en Ann Arbor, donde, dentro del grupo lógica de computadoras, sus ideas comenzaron a desarrollarse y a dar frutos.

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.



David Golberg

Figura 1.7

David Golberg, en la actualidad es el maestro de los algoritmos genéticos, conoció a Holland, y se convirtió en su estudiante y discípulo, aprendió su teoría y la aplicó mediante la computación evolutiva.

1.5.2. La evolución

La teoría de la evolución (que no es tal teoría, sino una serie de hechos probados), fue descrita por Charles Darwin veinte 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 cuál 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.

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 se les denominó alelos. En realidad, las teorías de Mendel, que trabajó en total aislamiento, se olvidaron y no se volvieron a redes cubrir 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.

Por la misma época, el biólogo alemán Walther Flemming describió los cromosomas, como ciertos filamentos en los que se agregaba la cromatina del núcleo celular durante la división; poco más adelante se descubrió que las células de cada especie viviente tenían un número fijo y característico de cromosomas. Y no fue hasta los años 50, cuando Watson y Crick descubrieron que la base molecular de los genes está en el ADN, ácido desoxirribonucleico. Los cromosomas están compuestos de ADN, y por tanto los genes están en los cromosomas.

La macromolécula de ADN está compuesta por bases púricas y pirimidínicas, la adenina, citosina, guanina y timina. La combinación y la secuencia de estas bases forma el código genético, único para cada ser vivo. Grupos de tres bases forman un codon, y cada codon codifica un aminoácido; el código genético codifica todas las proteínas que forman parte de un ser vivo. Mientras que al código genético se le llama genotipo, al cuerpo que construyen esas proteínas, modificado por la presión

ambiental, la historia vital, y otros mecanismos dentro del cromosoma, se llama gen.

No toda la cadena de ADN codifica proteínas, es decir, no todos son genes; las zonas que codifican proteínas se llaman intrones, las zonas que no lo hacen, exones. La cantidad de ADN basura aumenta desde los seres vivos más simples, como las bacterias, donde no hay nada, hasta los seres humanos, donde gran cantidad del ADN no codifica.

Un gen comienza con el sitio tres o aceptor y termina con el sitio cinco o donante. Proyectos como el del Genoma Humano tratan de identificar cuáles son estos genes, sus posiciones, y sus posibles alteraciones, que habitualmente conducen a enfermedades.

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 las poblaciones y dentro de las poblaciones: reproducción, mutación, competición y selección. La evolución se puede definir entonces como cambios en el conjunto genético de una población.

Un tema polémico, con opiniones variadas dependiendo de que se trate de informáticos evolutivos o de biólogos o geneticistas, 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; además, indicios de esta optimización se encuentran en el organismo de los animales, desde el tamaño y tasa de ramificación de las arterias, diseñada para maximizar flujo, hasta el metabolismo, que optimiza la cantidad de energía extraída de los alimentos.

Sin embargo, los geneticistas y biólogos evolutivos afirman que la evolución no optimiza, sino que adapta y optimiza localmente en el espacio y el tiempo; evolución no significa progreso. Un organismo más evolucionado puede estar en desventaja competitiva con uno de sus antepasados, si se colocan en el ambiente del último.

1.5.3. La informática evolutiva

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 autorreproducirse,

por tanto, un autómeta o máquina autorreproductiva tendría que ser capaz, aparte de contener las instrucciones para hacerlo, de copiar tales instrucciones a su descendencia, y así continuar con este proceso.

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 mejorar procesos industriales. La técnica de Box, denominada EVOP (Operación Evolucionaria), 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 hipercubo durante un tiempo, y al final del periodo de pruebas, un comité humano decidía sobre la calidad 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, Friedberg 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 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 aleles), 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 ceros y unos, 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, 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, y luego 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 al nivel de población, es decir que actúa de manera global.

El primer uso de procedimientos evolutivos en inteligencia artificial se debe 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. Un autómata celular es un conjunto de estados y reglas de transición entre ellos, de forma

que, al recibir una entrada, cambia o no de estado y produce una salida.

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 cual 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 esta 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 de mutación a los cromosomas existentes en la población, en cada generación, se elimina un porcentaje de la población (especificado por los parámetros), y los restantes generan la población total, mediante mutación y crossover. La magnitud del vector mutación se calcula adaptativamente.

John Holland desde pequeño, se preguntaba cómo logra la naturaleza, crear seres cada vez más perfectos. Lo curioso era que todo se lleva a cabo basándose en interacciones locales entre individuos, y entre estos y lo que les rodea. No sabía la respuesta, pero tenía una cierta idea de como hallarla: tratando de hacer pequeños modelos de la naturaleza, que tuvieran alguna de sus características, y ver cómo funcionaban, para luego extrapolar sus conclusiones a la totalidad. De hecho, ya de pequeño hacía simulaciones de batallas célebres con todos sus elementos: copiaba mapas y los cubría luego de pequeños ejércitos que se enfrentaban entre sí.

En los años 50 entró en contacto con 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 lógica de computadoras, sus ideas comenzaron a desarrollarse y a dar frutos. Además, fue leyendo un libro escrito por un biólogo evolucionista llamado, 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.

Por tanto, cuando Holland se enfrentó a los algoritmos genéticos, los objetivos de su investigación fueron dos:

- La imitación de los procesos adaptativos de los sistemas naturales, y

- El diseño sistemas artificiales (normalmente programas) que retengan los mecanismos importantes de los sistemas naturales.

Unos 15 años más adelante, David Goldberg, que en la actualidad es el maestro de los algoritmos genéticos, conoció a Holland, y se convirtió en su estudiante. Goldberg 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 suficiente aceptado para celebrar la primera conferencia en 1985, ICGA'85, la cual se sigue celebrando bianualmente.

1.6. Ubicación del problema

El problema consiste en desarrollar un algoritmo genético para la simulación del crecimiento de una población, basándose en los índices de natalidad, mortalidad, y las migraciones, y luego observar su comportamiento a través del tiempo.

Además se ha de optimizar el crecimiento de una población, a través del tiempo, mediante algoritmos genéticos, y finalmente se realizará una selección de un modelo más complejo, basándose en el paradigma de los algoritmos genéticos.

1.7. Limitaciones y alcance del tema.

Las técnicas de simulación convencionales, son de gran utilidad; pero aplicando los algoritmos genéticos, se puede realizar la simulación del crecimiento de una población demográfica, aplicando el proceso de selección natural y otros operadores genéticos, donde cada individuo forma parte de la población, y por lo tanto, pertenece a la solución.

1.8. Objetivos: generales y específicos.

El objetivo general es desarrollar un modelo de crecimiento poblacional, utilizando el paradigma de los algoritmos genéticos como una herramienta robusta para la simulación y la optimización y se tratará de estudiar el comportamiento de una población mediante un modelo basado en el proceso de la evolución.

Como objetivos específicos se tienen los siguientes:

-  Observar el comportamiento de los índices poblacionales, así como la posibilidad de una tendencia, o convergencia en un modelo desarrollado con algoritmos genéticos.
-  Realizar variaciones en los algoritmos genéticos de para adaptarlos al problema planteado.
-  Implantar una aplicación robusta que aplique los algoritmos genéticos para modelar el crecimiento de una población, y sus interacciones.
-  Comparar los modelos de crecimiento poblacional convencionales, con un modelo basado en algoritmos genéticos.

CAPITULO 2

2. Teoría de los algoritmos genéticos

2.1. Introducción

En este capítulo encontramos la base teórica y matemática de los algoritmos genéticos, es decir porqué funcionan, así como el algoritmo genético básico, y varias técnicas de representación para la programación de los algoritmos genéticos con todas sus posibles representaciones y sus operadores genéticos comunes y especiales.

Luego se realiza una explicación detallada del funcionamiento de las redes neuronales, autómatas celulares y los algoritmos meméticos, así como sus posibles aplicaciones, dando lugar a una comparación entre los algoritmos genéticos y otras técnicas de programación evolutiva.

2.2. Marco teórico

2.2.1. El teorema de esquemas

Este teorema proporciona el fundamento teórico de porqué los algoritmos genéticos pueden resolver diversos problemas, y porqué se adaptan a dichos problemas, acercándose rápidamente a una posible solución. En su análisis se considera el proceso de selección y los operadores de cruce y mutación.

Un esquema se construye utilizando un nuevo símbolo (*) para representar un comodín que puede aparear ambos valores, cero o uno (0 o 1). Por ejemplo, el esquema 11*00* representa las cadenas: 111001, 111000, 110001, 110000.

2.2.1.1. El orden de los esquemas

El orden de un esquema es el número de elementos que no son (*) dentro del esquema. Supongamos los siguientes esquemas:

$$Q_1 = (*01*),$$

$$Q_2 = (* ** 1),$$

$$Q_3 = (110*),$$

Si $o(Q_i)$ es el orden del esquema Q_i , se tiene los siguientes órdenes:

$$o(Q_1) = 2,$$

$$o(Q_2) = 1,$$

$$o(Q_3) = 3.$$

El concepto de orden del esquema se utiliza para calcular la probabilidad de supervivencia del esquema con relación al operador mutación.

2.2.1.2. La longitud de los esquemas

La longitud que define a un esquema es la distancia entre la primera posición fija y la última posición fija, considerando como posiciones fijas a aquellas que contienen un elemento diferente de (*). La longitud del esquema Q_i , se denota por medio de $\acute{o}(Q_i)$,

$$\acute{o}(Q_1) = 3 - 2 = 1$$

$$\acute{o}(Q_2) = 4 - 4 = 0$$

$$\acute{o}(Q_3) = 3 - 1 = 2.$$

El concepto anterior se utiliza para calcular la probabilidad de supervivencia del esquema frente al operador de cruce.

El teorema dice que: *“Los esquemas pequeños de bajo orden arriba del promedio reciben un incremento exponencial de*

representantes en las siguientes generaciones de un algoritmo genético". [Holland 92].

Entonces, supongamos un alfabeto binario $S = \{0, 1\}$. Para construir un esquema necesitamos ampliar el alfabeto anterior por medio del símbolo *. Un esquema será cualquier ristra de caracteres formada a partir de elementos del alfabeto ampliado S' , siendo $S' = \{0, 1, *\}$. Si la longitud de la ristra es l , el número de esquemas existentes es 3^l , ya que cada posición puede ocuparse por cualquier de los tres elementos del alfabeto extendido S' . Un esquema representa por tanto todas aquellas ristas que se asocian con él, en todas las posiciones distintas de las ocupadas por él.

Ejemplo:

Si se considera ristas de longitud 4, entonces se tiene que el esquema:

$$Q = (* 1 * 0)$$

se empareja con los cromosomas:

(0100),
(0110),
(1100),
(1110).

Desde un punto de vista geométrico un esquema equivale a un hiperplano en el espacio de búsqueda.

Ahora veamos algunos conceptos que son necesarios para comprender el Teorema:

- **n** : Numero total de individuos de una población.
- **$f(h)$** : Valor de la función de adaptación para un individuo concreto denominado h .
- **$f(t)$** : Valor medio de la función de adaptación de todos los individuos de una población en una generación t .
- **$p(t)$** : Representa la población en una generación t .
- **$h \in p(t)$** : Expresión que indica que un individuo h pertenece tanto a la población en un instante de tiempo como al conjunto de individuos representados por el esquema s .
- **$\bar{u}(s,t)$** : Valor medio de la función de adaptación de los individuos del esquema s en la generación t .
- **$m(s,t)$** : Numero de individuos del esquema s en la generación t .
- **$m(s,t+1)$** : Numero de individuos del esquema s en la siguiente generación.
- **$E[m(s,t+1)]$** : Significa lo mismo que la expresión anterior.

- **$\Pr(h)=f(h)/nf(t)$** : Probabilidad de que un individuo sea seleccionado en la generación t
- **$\Pr(h\hat{=} s)=\hat{u}(s,t)m(s,t)/nf(t)$** : Probabilidad de que un individuo del esquema s sea seleccionado
- **$E[m(s,t+1)]=\hat{u}(s,t)m(s,t)/f(t)$** : Considerando solamente el efecto de la selección sobre la población, esta expresión nos da el valor del numero de individuos del esquema s en la siguiente generación, que será la probabilidad de que seleccionemos un individuo por el número de individuos totales de la población.

El análisis hecho arriba considera sólo el paso de selección del algoritmo genético, la combinación y la mutación deben ser considerados también. El Teorema fundamental considera únicamente la posibilidad de influencia negativa de estos operadores genéticos. El Teorema completo se enuncia a continuación, tras haber definido antes algunos conceptos:

- **L** : Longitud de un individuo.
- **pc** : Probabilidad de que el operador de combinación se aplique a un individuo.
- **pm** : Probabilidad de que un bit de un individuo sea mutado.
- **$o(s)$** : Numero de bits definidos en el esquema s .

- **d(s):** Distancia máxima entre dos bits definidos en el esquema s.

$$E[m(s, t+1)] \geq \left[\frac{\hat{u}(s, t) m(s, t)}{f(t)} \right] \cdot \left[1 - \frac{p_c \cdot d(s)}{(L-1)} \right] \cdot \left[(1 - p_m)^{o(s)} \right]$$

El primer factor de la expresión representa, como ya hemos visto, el efecto del operador de selección en la siguiente generación. El factor medio y último describen el efecto de los operadores de combinación (de un solo bit) y mutación, en particular describen la probabilidad de que un individuo concreto que es representado por el esquema s pueda seguir siendo representado por dicho esquema tras haber sido aplicado sobre el operador de combinación y mutación respectivamente.

La fórmula anterior nos proporciona una cota inferior del número esperado de individuos que se asocian con el esquema Q en la siguiente generación, expresado en función del número de individuos que se asocian con el esquema en la actual generación, así como de la adaptación del esquema relativa a la adaptación media de la población, el orden y la longitud del esquema, la longitud de las ristas, así como las probabilidades de cruce y mutación. Tal y como se desprende de la fórmula anterior, esquemas cortos, de bajo orden y con una adaptación al problema

superior a la adaptación media, se espera que a medida que evoluciona el Algoritmo Genético, obtengan un incremento exponencial en el número de individuos que se asocian con los mismos.

2.2.2. Paralelismo implícito

El hecho de que los algoritmos genéticos efectúen búsquedas robustas, se debe a que implícitamente muestrean hiperplanos. Es decir cada individuo, representado por medio de una codificación binaria, constituye un vértice del hipercubo que representa el espacio de búsqueda, y es a su vez un miembro de $2^l - 1$ diferentes hiperplanos, donde l es la longitud de la codificación binaria. Por otra parte existen un total de $3^l - 1$ hiperplanos particiones sobre el espacio de búsqueda total.

Cada individuo no proporciona gran información si se examina por separado, es por ello que la búsqueda basada en una población es crítica para los algoritmos genéticos. Una población de individuos, proporciona información sobre numerosos hiperplanos, estando los hiperplanos de bajo orden hechos un muestreo por numerosos puntos de la población. El hecho de que se muestreen muchos hiperplanos cuando se evalúa una población de ristas, se conoce como el paralelismo implícito o intrínseco de los algoritmos

genéticos. Dicho paralelismo implícito, significa que muchas competiciones entre hiperplanos se efectúan simultáneamente.

2.2.3. Teoremas de convergencia

Dado que el algoritmo genético opera con una población en cada iteración, se espera que el método converja de modo que al final del proceso la población sea muy similar, y en el infinito se reduzca a un sólo individuo.

Se ha desarrollado toda una teoría para estudiar la convergencia de estos algoritmos en el caso de cadenas binarias. Esta teoría se basa principalmente en considerar que una cadena es un representante de una clase de equivalencia o esquema, reinterpretando la búsqueda en lugar de entre cadenas, entre esquemas.

A partir de este resultado el teorema de esquemas (o teorema fundamental), prueba que la población converge a unos esquemas que cada vez son más parecidos, y en el límite a una única cadena.

En el caso de cadenas no binarios se introducen los conceptos de forma y conjunto de similitud que generalizan al de esquema. Se consideran una serie de condiciones sobre los operadores de

manera que se garantice la convergencia. Básicamente se exige que al cruzar dos cadenas de la misma clase se obtenga otro dentro de ésta. Además hay que respetar ciertas condiciones sobre selección de los progenitores. Bajo toda esta serie de hipótesis se prueba la convergencia del algoritmo.

En la práctica no se suelen respetar las condiciones vistas ya que son difíciles de seguir y probar, encontrándonos con que, en ocasiones los algoritmos genéticos resuelven satisfactoriamente un problema de optimización dado y otras se quedan muy alejadas del óptimo. Los estudiosos del tema han tratado de caracterizar lo que han denominado problemas algoritmos genéticos fáciles (aquellos en los que los algoritmos genéticos proporcionan buenos resultados) y algoritmos genéticos difíciles con el objetivo de saber de antemano, al estudiar un nuevo problema, si los algoritmos genéticos son una buena elección para su resolución.

Se han tratado de caracterizar estas clases mediante el concepto de engaño considerando que si el algoritmo converge al mejor esquema (aquel con mejor promedio de aptitud de sus cadenas) y en éste se encuentra el óptimo, entonces es fácil que se resuelva satisfactoriamente. En caso de que el óptimo esté en un esquema con bajo promedio se denomina engaño y se pensaba que en

estos casos es cuando el problema es Algoritmo genético difícil. Sin embargo se ha visto que esta caracterización mediante el engaño no es siempre cierta y no constituye un criterio fiable.

Rudolph demuestra la no convergencia hacia el óptimo global del algoritmo genético canónico, así como la garantía, de convergencia expresada en términos probabilísticos, del algoritmo genético que mantiene a la mejor solución en la población.

Davis y Principe extrapolan los fundamentos teóricos del algoritmo simulado de alineación a un modelo de algoritmo genético basado en cadenas de Markov. Se efectúa un estudio de las matrices de transición de estados teniendo en cuenta en primer lugar tan sólo la reproducción, a continuación la reproducción y la, mutación y finalmente la reproducción, la mutación y el cruce.

Suzuki efectúa un estudio de la convergencia de los algoritmos genéticos por medio de cadenas de Markov. Los algoritmos genéticos estudiados presentan un criterio de reducción elitista modificado, según el cual se genera una población de A individuos, incluyendo en ella al mejor individuo de la población en la generación anterior, obteniéndose los $A - 1$ individuos restantes por medio de las operaciones genéticas normales.

Liepins demuestra la convergencia del algoritmo genético hacia poblaciones que contienen al óptimo, en el caso de algoritmos genéticos sin operador de mutación; pero en los cuales el reemplazo de individuos es elitista - el mejor individuo no se pierde nunca – y además se efectúa de tal manera que en cada paso cualquier punto del espacio sea potencialmente alcanzable por medio de la operación de cruce.

Chakraborty y Dastidar presentan un modelo de fiabilidad estocástica de un esquema para el algoritmo genético binario con longitud de representación fija, y obtienen una estimación para el número de generaciones necesarias hasta obtener la convergencia.

Eiben y Col modelan la evolución del algoritmo genético por medio de una cadena de Markov, obteniendo condiciones suficientes para la convergencia en probabilidad del proceso evolutivo hacia el óptimo global.

2.3. La estructura de un algoritmo genético

Un algoritmo genético está descrito por tres módulos de programación:

- a) **Modulo evolutivo**, Es un mecanismo de decodificación que interpreta la información de un cromosoma y tiene una función de evaluación que mide la calidad del cromosoma. Este es el único punto donde existe información del dominio.
- b) **Modulo poblacional**, Este módulo tiene una representación poblacional y técnicas para manipularla. Se utilizan técnicas de representación, técnicas de arranque, criterios de selección y de reemplazo. Aquí también se define el tamaño de la población y la condición de terminación.
- c) **Modulo reproductivo**, Este módulo contiene los operadores genéticos así como lo son el crossover, la mutación la selección, entre otros.

2.4. Tipos de algoritmos genéticos

Los algoritmos genéticos se han aplicado a diversos problemas dentro del aprendizaje automático solucionando tareas que van de la planificación al desarrollo de topologías neuronales. De entre todas estas aplicaciones la que ha sido, a nuestro entender, más prometedora es la del aprendizaje supervisado, es decir, a partir de una serie de ejemplos (y posiblemente contraejemplos) de uno o varios conceptos, generar un conjunto de reglas que permitan,

dado un individuo, decidir la clase a la cual pertenece. Las soluciones propuestas para resolver este tipo de problemas pueden caracterizarse básicamente según dos criterios:

➤ **¿Qué representa un individuo en la población?**

En general ha habido dos respuestas a la primera de las preguntas, cada una de las cuales representa una escuela diferente en el uso de algoritmos genéticos de aprendizaje. Dichas escuelas son:

Michigan: En este enfoque los individuos de la población representan reglas individuales, por lo que la solución del problema consiste en toda la población.

Pittsburgh: La población consiste en conjuntos de reglas de cardinalidad variable, por tanto, cada individuo representa una posible solución del problema a resolver.

Dentro del primer enfoque cabe destacar los sistemas clasificadores, cuyo estudio y análisis por si mismo constituye una disciplina aparte. Todos los sistemas que siguen este enfoque tienen un problema adicional cuya resolución es crítica para su funcionamiento: como la solución del problema es toda la población ¿cómo se evalúa la bondad de sus individuos?, pues la bondad de la solución hace referencia a todos ellos.

Los sistemas que siguen la segunda tendencia siguen muy fielmente el esquema clásico de algoritmo genético ya que la población consiste en diferentes soluciones posibles que compiten entre sí.

➤ **¿Cómo se representan los objetos?**

Existen básicamente dos tipos de representaciones, las cuales a su vez influyen en la definición de los operadores genéticos a utilizar.

Clásicas: Se mantiene la representación de cadenas binarias, aunque algunas veces de longitud variable.

Ad hoc: En este caso la representación es mucho más cercana al problema a resolver, y por ejemplo, en tareas de clasificación suele utilizarse la representación atributo valor.

En el primer caso, que podría denominarse minimalista, no deben hacerse apenas cambios sobre un algoritmo genético estándar. El escollo principal consiste en encontrar una representación binaria adecuada a nuestro problema. La gran ventaja radica en que la mayor parte de la teoría de los algoritmos genéticos se ha realizado para este tipo de representaciones.

Para el otro tipo de representaciones se deben diseñar también los operadores genéticos de forma particular (lo cual puede ser

complicado) pero, al estar mucho más cercanos al problema a resolver, suelen ser más eficientes. Además, al utilizar representaciones de más alto nivel, la hibridación con otro tipo de sistema de aprendizaje es mucho más simple.

Existen varios tipos de algoritmos genéticos, cada uno basado en una metáfora distinta de la naturaleza:

- Algoritmos genéticos generacionales.
- Algoritmos genéticos de estado fijo.
- Algoritmos genéticos paralelos.

2.4.1. Algoritmos genéticos generacionales

Se asemeja a la forma de reproducción de los insectos, donde una generación pone huevos, se aleja geográficamente o muere y es substituida por una nueva. En este modelo se realizan cruces en una piscina de individuos, los descendientes son puestos en otra, al final de la fase reproductora se elimina la generación anterior y se pasa a utilizar la nueva. Este modelo también es conocido como algoritmo genético canónico.

2.4.2. Algoritmos genéticos de estado fijo

Utilizan el esquema generacional de los mamíferos y otros animales de vida larga, donde coexisten padres y sus

descendientes, permitiendo que los hijos sean educados por sus progenitores, pero también que a la larga se genere competencia entre ellos.

En este modelo, no sólo se debe seleccionar los dos individuos a ser padres, si no también cuales de la población anterior serán eliminados, para hacer espacio a los descendientes.

La diferencia esencial entre el modelo de reemplazo generacional y el modelo de estado fijo es que las estadísticas de la población son recalculadas luego de cada cruce y los nuevos descendientes están disponibles inmediatamente para la reproducción. Esto permite al modelo utilizar las características de un individuo prometedor tan pronto como es creado.

Algunos autores dicen que este modelo tiende a evolucionar mucho más rápido que el modelo generacional [BD93], sin embargo entre las investigaciones de Goldberg y Deb (1993) se encontró que las ventajas parecen estar relacionadas con la alta tasa de crecimiento inicial, ellos dicen que los mismos efectos pueden ser obtenidos en rangos de adaptación exponencial o selección por competencia. No encontraron evidencia que este modelo sea mejor que el generacional.

2.4.3. Algoritmos genéticos paralelos

Estos algoritmos son denominados también algoritmos meméticos. Parte de la metáfora biológica que motivó a utilizar la búsqueda genética consiste en que es inherentemente paralela, donde al evolucionar se recorren simultáneamente muchas soluciones, cada una representada por un individuo de la población. Sin embargo, es muy común en la naturaleza que no sólo sea una población evolucionando, si no varias poblaciones, normalmente aisladas geográficamente, que originan respuestas diferentes a la presión evolutiva. Esto origina dos modelos que toman en cuenta esta variación, y utilizan no una población como los anteriores sino múltiples concurrentemente.

Una tendencia relativamente nueva en el campo de los algoritmos genéticos es la incorporación de técnicas de búsqueda local. Esta aplicación tras una increíble mejora en algunos resultados que fueron desalentadores utilizando los algoritmos genéticos cuando han sido aplicados en problemas específicos, especialmente en los más difíciles de optimización combinatoria, por esta razón los investigadores comenzaron a buscar nuevas maneras para extender los algoritmos genéticos y obtener mejores resultados en períodos de tiempo menores.

Un paso importante en esta dirección fue dado por Muhlenbein con un desenvolvimiento de algoritmos genéticos paralelos (PGA's) que permiten que los individuos en una población mejoren sus disposiciones mediante mejoras iterativas (conocidas como alpinismo de la colina). Los algoritmos genéticos paralelos también permiten que los individuos se seleccionen entre ellos por procesos locales, o que faciliten su implantación en sistemas locales.

De cierto modo, los algoritmos genéticos simples con búsqueda local pueden ser vistos cómo métodos sofisticados de descenso múltiple. El proceso de reiniciación está en este caso comandado por reglas genéticas, y la fase de descenso está encerrada cómo de traje. El éxito de los métodos puede ser atribuido a su equilibrio entre tener una búsqueda rápida y mantener una diversidad para evitar la convergencia prematura. Otro método puede consistir en desenvolver una búsqueda iterativa llamada codificación Delta propuesta por D. Whitley y sus colaboradores. La codificación Delta introdujo diversidad, generando aleatoriamente una nueva población, no en tanto preserva la información de las generaciones previas basándose en las nuevas codificaciones y en soluciones parciales previas.

2.4.3.1. Modelos de islas

Si se tiene una población de individuos, esta se divide en subpoblaciones que evolucionan independientemente como un algoritmo genético normal. Ocasionalmente, se producen migraciones entre ellas, permitiéndoles intercambiar material genético.

Con la utilización de la migración, este modelo puede explotar las diferencias en las subpoblaciones; esta variación representa una fuente de diversidad genética. Sin embargo, si emigra un gran número de individuos en cada generación, ocurre una mezcla global y se eliminan las diferencias locales, y si la migración es infrecuente, es probable que se produzca convergencia prematura en las subpoblaciones.

2.4.3.2. Modelo celular

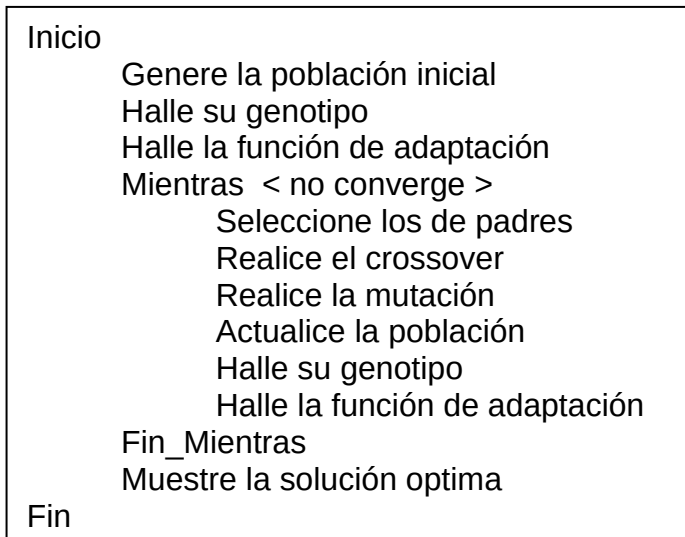
Coloca cada individuo en una matriz, donde cada uno sólo podrá buscar reproducirse con los individuos que tenga a su alrededor (más cerca de casa) escogiendo al azar o al mejor adaptado. El descendiente pasará a ocupar una posición cercana.

No hay islas en este modelo, pero hay efectos potenciales similares. Asumiendo que el cruce está restringido a individuos

adyacentes, dos individuos separados por 20 espacios están tan aislados como si estuvieran en dos islas, este tipo de separación es conocido como aislamiento por distancia.

Luego de la primera evaluación, los individuos están todavía distribuidos al azar sobre la matriz. Posteriormente, empiezan a emerger zonas con cromosomas y adaptaciones semejantes. La reproducción y selección local crea tendencias evolutivas aisladas, luego de varias generaciones, la competencia local resultará en grupos más grandes de individuos semejantes.

2.5. El algoritmo genético básico



El algoritmo genético básico

Cuadro 2.1

El algoritmo es realmente sencillo y fácil de comprender, y debe recordarse que la parte más importante se desea llegar a una solución es la función de adaptación que se escoja, ya que de esta dependerá también una correcta selección de padres.

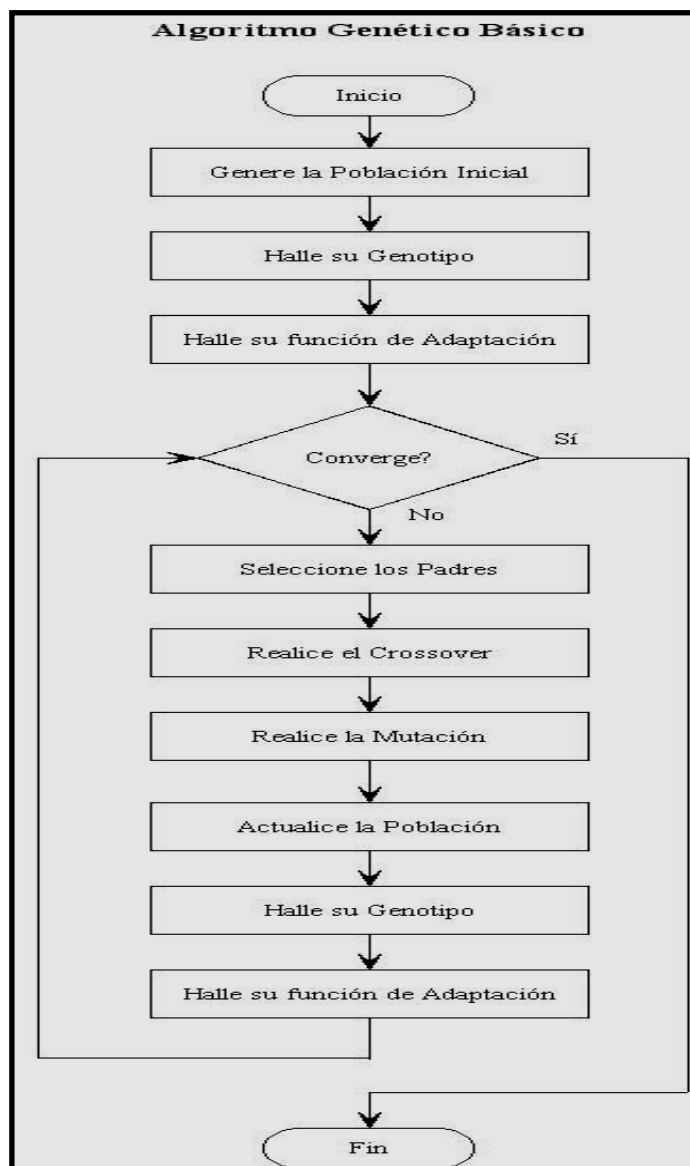


Diagrama de flujo del algoritmo genético básico

Figura 2.1

2.5.1. La población inicial

La población inicial consta de n individuos, que conforman una solución inicial. Por lo general esta población es generada mediante cadenas de símbolos o caracteres, para luego ser procesados.

Los tipos más comunes de codificación son:

- Codificación Binaria
- Codificación por permutaciones
- Codificación por símbolos
- Codificación mediante árboles

2.5.1.1. Codificación binaria

La codificación en base binaria es la más común de todas, ya que luego es sencillo llevarlo a base decimal, para obtener una función de adaptación.

Evidentemente los cromosomas estarán formados por cadenas de ceros y unos. Esto permitirá representar un intervalo $[a, b)$, en donde se tomarán n puntos que serán números binarios de una longitud ideal de logaritmo en base dos de n ($\log_2 n$).

Cromosoma A	101100101100101011100101
Cromosoma B	111111100000110000011111

Ejemplo de dos cromosomas en codificación binaria

Cuadro 2.2

Estos cromosomas tienen una longitud 24, por lo que con ellos se puede representar 2^{24} números, es decir 16'777.216 números.

2.5.1.2. Codificación por permutaciones

Este tipo de codificación puede ser utilizado en problemas de ordenamiento, y sus números representan una secuencia.

Cromosoma A	(1 5 3 2 6 4 7 9 8)
Cromosoma B	(8 5 6 7 2 3 1 4 9)

Ejemplo de dos cromosomas en la representación por permutaciones

Cuadro 2.3

Un problema en el que se puede aplicar esta técnica de codificación es el TSP, que consiste en la optimización de una ruta.

2.5.1.3. Codificación por valores

En este caso se atribuyen diferentes valores a cada cromosoma, que pueden ser reales, alfabéticos, o símbolos especiales.

Cromosoma A	1.2324 5.3243 0.4556 2.3293 2.4545
Cromosoma B	ABDJEIFJDHDIERJFDLDFLFEGT
Cromosoma C	(abajo), (abajo), (derecha), (arriba), (izquierda)

Ejemplo de tres cromosomas en codificación por valores

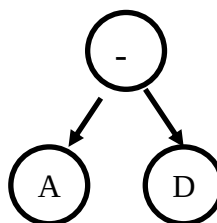
Cuadro 2.4

En realidad este tipo de codificación suele ser bastante buena en algunos problemas.

2.5.1.4. Codificación mediante árboles

En este tipo de codificación cada árbol representará un cromosoma, y las ventajas es que con un buen manejo de árboles, la programación se facilitará muchísimo.

Por ejemplo:



Ejemplo de un cromosoma de la codificación por árboles

Figura 2.2

2.5.2. El genotipo

El genotipo de cada cromosoma, representa el valor decimal de la cadena de símbolos o caracteres generados. Para cierto tipo de problemas, este genotipo debe ser llevado a un intervalo $[a, b)$ mediante un pequeño cambio de variable, de tal manera que se representen solamente los valores que pertenecen a dicho intervalo.

2.5.3. La función de adaptación.

Una gran ventaja de los algoritmos genéticos sobre las otras técnicas de optimización es que la función de adaptación puede ser la misma función que deseamos optimizar, y no se necesita utilizar ningún otro conocimiento matemático tal como derivadas, inversas, o cualquier tipo de artificio para luego tener una adaptación al problema.

Dos aspectos que resultan cruciales en el comportamiento de los algoritmos genéticos son la determinación de una adecuada función de adaptación o función objetivo, así como la codificación utilizada.

La función de adaptación es de muchísima importancia para el algoritmo genético, ya que de esta depende que el problema

pueda o no resolverse, y aunque parece sencillo manipular una expresión que será la misma que se ha planteado, se presentará un inconveniente cuando no se pueda utilizar una función de adaptación, porque esta no se conoce.

Idealmente nos interesaría construir funciones objetivo con ciertas regularidades, es decir funciones objetivo que verifiquen que para dos individuos que se encuentren cercanos en el espacio de búsqueda, sus respectivos valores en las funciones objetivo sean similares. Por otra parte una dificultad en el comportamiento del algoritmo genético puede ser la existencia de gran cantidad de óptimos locales, así como el hecho de que el óptimo global se encuentre muy aislado.

La regla, general para construir una buena función objetivo es que ésta debe reflejar el valor del individuo de una manera real; pero en muchos problemas de optimización combinatoria, donde existe gran cantidad de restricciones, buena parte de los puntos del espacio de búsqueda representan individuos no válidos.

Para este planteamiento en el que los individuos están sometidos a restricciones, se han propuesto varias soluciones. La primera sería la que se podría denominar absolutista, en la que aquellos individuos que no verifican las restricciones, no son considerados

como tales, y se siguen efectuando cruces y mutaciones hasta obtener individuos válidos, o bien, a dichos individuos se les asigna una función objetivo igual a cero.

Otra posibilidad consiste en reconstruir aquellos individuos que no verifican las restricciones. Dicha reconstrucción suele llevarse a cabo por medio de un nuevo operador que se acostumbra a denominar reparador.

Otro enfoque está basado en la penalización de la función objetivo. La idea general consiste en dividir la función objetivo del individuo por una cantidad (la penalización) que guarda relación con las restricciones que dicho individuo viola. Dicha cantidad puede simplemente tener en cuenta el número de restricciones violadas ó bien el denominado costo esperado de reconstrucción, es decir el coste asociado a la conversión de dicho individuo en otro que no viole ninguna restricción.

Otra técnica que se ha venido utilizando en el caso en que la computación de la función objetivo sea muy compleja es la denominada evaluación aproximada de la función objetivo. En algunos casos la obtención de n funciones objetivo aproximadas puede resultar mejor que la evaluación exacta de una única función objetivo (supuesto el caso de que la evaluación

aproximada resulta como mínimo n veces más rápida que la, evaluación exacta).

Un problema habitual en las ejecuciones de los algoritmos genéticos surge debido a la velocidad con la que el algoritmo converge. En algunos casos la convergencia es muy rápida, lo que suele denominarse convergencia prematura, en la cual el algoritmo converge hacia óptimos locales, mientras que en otros casos el problema es justo el contrario, es decir se produce una convergencia lenta del algoritmo. Una posible solución a estos problemas pasa por efectuar transformaciones en la función objetivo. El problema de la convergencia prematura, surge a menudo cuando la selección de individuos se realiza de manera proporcional a su función objetivo. En tal caso, pueden existir individuos con una adaptación al problema muy superior al resto, que a medida que avanza el algoritmo dominan a la población. Por medio de una transformación de la función objetivo, en este caso una comprensión del rango de variación de la función objetivo, se pretende que dichos superindividuos no lleguen a dominar a la población.

El problema de la lenta convergencia del algoritmo, se resolvería de manera análoga, pero en este caso efectuando una expansión del rango de la función objetivo.

La idea de especies de organismos, ha sido imitada en el diseño de los algoritmos genéticos en un método propuesto por Goldberg y Richardson, utilizando una modificación de la función objetivo de cada individuo, de tal manera que individuos que estén muy cercanos entre sí devalúen su función objetivo, con objeto de que la población gane en diversidad.

2.5.4. La selección de padres

La selección de padres es un proceso donde se escogerá mediante algún método los padres más aptos, simulando la ley de supervivencia de los individuos más aptos.

Este proceso es de gran importancia, ya que de una buena selección de padres, bajo un criterio lo suficientemente bueno, se realizará una convergencia más rápida, hacia una solución más óptima.

Los tipos más comunes de métodos o funciones de selección son:

- Modelo de selección proporcional a la función objetivo.
- Modelo de selección proporcional al rango.

- Modelo de selección del valor esperado.
- Modelo de muestreo con remplazo del resto.
- Modelo universal estocástico.
- Modelo de la rueda de la ruleta.
- Modelo del Torneo.
- Modelo de mantener el estado.

2.5.4.1. Modelo de la selección proporcional a la función objetivo.

En esta función cada individuo tiene una, probabilidad de ser seleccionado como padre que es proporcional al valor de su función objetivo.

Denotemos por $P(j)$ la probabilidad de que el individuo j sea seleccionado como padre y llamemos $g(j)$ a la función de adaptación del problema para cualquier individuo j . Entonces podemos decir que:

$$P(j) = \frac{g(j)}{\sum_{j=1}^n g(j)}$$

Esta función de selección es invariante ante un cambio de escala, pero no ante una traslación.

2.5.4.2. Modelo de la selección proporcional al rango.

Se produce una repartición más uniforme de la probabilidad de selección. Denotemos por $R(g(j))$ al rango de la función objetivo para el individuo j . Entonces se puede decir que:

$$P(j) = \frac{R(g(j))}{\frac{n(n+1)}{2}}$$

Donde el denominador es la suma de los rangos de todos los individuos. La función de selección basada en el rango es invariante frente a la translación y al cambio de escala.

2.5.4.3. Modelo de la selección del valor esperado.

Para cada individuo SI , se introduce un contador, inicializado en $g(j)/g_t$, donde, g_t denota la media, de la función objetivo en la generación t . Cada vez que el individuo (j) es seleccionado para el cruce, dicho contador decrece en una cantidad c (c pertenece a $(0, 5; .1)$). El individuo en cuestión dejará de poder ser seleccionado en esa generación, cuando su contador sea negativo.

2.5.4.4. Modelo de muestreo con remplazo del resto.

Cada individuo es seleccionado un número de veces que coincide con la parte entera del número esperado de ocurrencias de dicho suceso, compitiendo los individuos por los restos. Es decir que si denotamos por $m(j)$ el número de veces que el individuo j es seleccionado para el cruce, tenemos que:

$$m(j) = m_1(j) + m_2(j)$$

$$m_1(j) = g(j) / gt$$

$$m_2(j) = \text{muestrear sobre } n - \sum_{j=1}^n \frac{g(j)}{gt}$$

2.5.4.5. Modelo del muestreo universal estocástico.

Efectuando un paralelismo con los métodos de muestreo estadísticos, este último tipo de selección de padres se relaciona con el muestreo sistemático, mientras que la selección proporcional a la función objetivo, está basada en el muestreo estratificado con fijación proporcional al tamaño. También el procedimiento de selección que hemos denominado muestreo estocástico con reemplazo del resto, mantiene un paralelismo con el muestreo estratificado con fijación de compromiso.

2.5.4.6. Modelo de la rueda de la ruleta.

Es un método muy utilizado, ya que puede realizárselo mediante pocos pasos muy fáciles de comprender y aplicar:

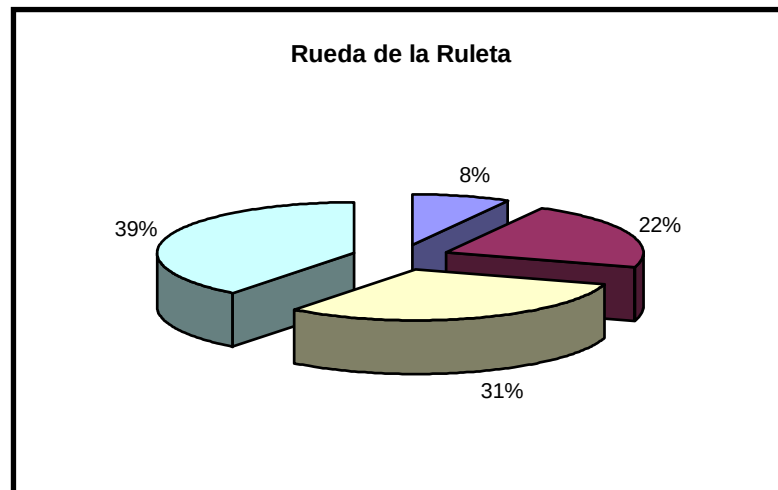


Gráfico de la selección de la rueda de la ruleta

Figura 2.3

INICIO
a. Halle la suma de las funciones de aptitud
b. Halle la frecuencia relativa a la suma
c. Halle la frecuencia acumulada
d. Genere un número aleatorio con distribución $U(0,1)$
e. $l=1$
f. Si el Número es menor que la l -ésima frecuencia acumulada, entonces seleccionamos el l -ésimo cromosoma.
Ir a i.
g. $l=l+1$
h. Ir a f.
i. FIN

Algoritmo para realizar el método de selección de la rueda de la ruleta

Cuadro 2.5

El problema es en cuanto a la disposición cuando ésta difiere mucho. Por ejemplo, si la aptitud del cromosoma mejor es 90% entonces de toda la rueda de la ruleta los otros cromosomas tendrán muy pocas oportunidades ser seleccionado.

2.5.4.7. Modelo del torneo

Se asigna un rango a las líneas de la población y entonces cada cromosoma recibe aptitud de esta clasificación jerárquica. El peor tendrá aptitud 1, segundo peor 2, etc. y los más buenos tendrán aptitud N (el número de cromosomas en población).

Después de esto todos los cromosomas tienen una oportunidad similar de ser seleccionados. Pero este método puede llevar a la convergencia más lenta, porque los cromosomas más buenos no difieren tanto de otro.

2.5.4.8. Modelo de mantener el estado.

Éste no es un método particular de seleccionar a los padres. La idea principal de esta selección es que una gran parte de cromosomas debe sobrevivir a la próxima generación.

Los algoritmos genéticos entonces trabajan de la siguiente manera. En cada generación se selecciona unos (bueno - con aptitud alta)

los cromosomas por crear una nueva descendencia. Entonces algunos (malo - con aptitud baja) los cromosomas están alejados y la nueva descendencia se pone en su lugar. El resto de población sobrevive a la nueva generación.

2.5.4.9. Elitismo

La idea de elitismo ya se ha introducido. Al crear nueva población por crossover y mutación, nosotros tenemos una gran oportunidad, de soltar el mejor cromosoma.

Elitismo es nombre del método que primero copia el mejor cromosoma (o unos cromosomas más buenos) a la nueva población. El resto se hace de manera clásica.

El elitismo encausa aumento de eficiencia del Algoritmo Genético, porque previene la pérdida de la solución que se encontró.

Una posible clasificación de procedimientos de selección de padres consistirá en: métodos de selección dinámicos, en los cuales las probabilidades de selección varían de generación a generación, (por ejemplo la selección proporcional a la función objetivo), frente a métodos de selección estáticos, en los cuales dichas probabilidades permanecen constantes (por ejemplo la selección basada en rangos).

Si se asegura que todos los individuos tienen asignada una probabilidad de selección distinta de cero el método de selección se denomina preservativo. En caso contrario se acostumbra a denominarlo extintivo.

2.6. Crossover

El crossover y la mutación son dos operadores básicos de los algoritmos genéticos. La actuación de los algoritmos genéticos depende mucho de ellos. El tipo y la aplicación de estos operadores dependen de la codificación en un problema.

2.6.1. Crossover de la codificación Binaria

Las formas más conocidas de crossover para codificación binaria son:

- Crossover en un solo punto.
- Crossover en n puntos.
- Crossover Uniforme.
- Crossover matemático.
- Máscara de crossover

2.6.1.1. Crossover en un punto

	Cromosoma 1	Cromosoma 2
Padres	11001010	10011011
Hijos	11001011	1001 1010

Ejemplo del crossover en un punto de la representación binaria

Cuadro 2.6

En este tipo de Crossover se elige aleatoriamente un punto, y se toman los bits desde el inicio hasta dicho punto de un cromosoma, y los demás del otro cromosoma. El crossover en un punto puede generalizarse a n puntos.

2.6.1.2. Crossover Uniforme

	Cromosoma 1	Cromosoma 2
Padres	11001010	10011011
Hijos	10001011	11011010

Ejemplo del crossover uniforme en la representación binaria

Cuadro 2.7

En este Crossover cada bit recibe igual probabilidad de ser seleccionado de los padres en cualquiera de los dos hijos, de manera similar a utilizar una máscara de crossover.

2.6.1.3. Crossover matemático

En el siguiente ejemplo de crossover matemático, el primer hijo nace a partir del operador de conjunción ($\wedge$), mientras que el segundo hijo nace a partir del operador de disyunción ($\vee$).

	Cromosoma 1	Cromosoma 2
Padres	11001010	10011011
Hijos	10001011	11011010

Ejemplo de crossover matemático en la representación binaria

Cuadro 2.8

En este tipo de Crossover, cada hijo será el resultado de una operación de lógica matemática, la que puede ser: $\wedge, \vee, \Rightarrow, \Leftrightarrow, \Leftarrow$.

2.6.1.4. Máscara de crossover.

Cuando se utiliza el denominado crossover binario uniforme, se puede trabajar con una máscara de crossover, la cual atribuye de antemano las posiciones a tomar para la descendencia, es decir, que la máscara de crossover también será una cadena binaria, con la condición que si en esta aparece el valor de uno, entonces el gen se tomará del primer padre, mientras que si aparece el valor de cero, el gen se tomará del segundo padre.

2.6.2. Crossover de la codificación por permutaciones

Básicamente se trabaja con el crossover en un solo punto, aunque podría ser diferente que en la representación binaria, la selección de la segunda parte del cromosoma.

2.6.2.1. Crossover en un solo punto

Se selecciona un punto del crossover, entonces hacemos que hasta este punto de la permutación se copie del primer padre, mientras que el segundo padre se examina y si el número no está todavía en la descendencia que se agrega.

Por ejemplo:

$$(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9) + (4\ 5\ 3\ 6\ 8\ 9\ 7\ 2\ 1) = (1\ 2\ 3\ 4\ 5\ 6\ 8\ 9\ 7)$$

Ejemplo de crossover en un solo punto de la representación por permutaciones

Cuadro 2.9

2.6.3. Crossover de la codificación por valores

Para este tipo de Crossover, pueden utilizarse todos los métodos del crossover binario, es decir crossover en un punto, crossover en n puntos o el crossover uniforme.

2.6.3.1. Crossover en un solo punto

	Cromosoma 1	Cromosoma 2
Padres	(1.29 5.68 2.86 4.11 5.55)	(1.32 5.24 2.75 4.22 5.55)
Hijos	(1.29 5.68 2.86 4.22 5.55)	(1.32 5.24 2.75 4.11 5.55)

Ejemplo de crossover en un solo punto de la representación por valores

Cuadro 2.10

En este tipo de Crossover se elige aleatoriamente un punto, y se toman los bits desde el inicio hasta dicho punto de un cromosoma, y los demás del otro cromosoma.

2.6.3.2. Crossover en n puntos

Este Crossover es una generalización del Crossover en un solo punto, se selecciona n posiciones para realizar el intercambio de bits.

	Cromosoma 1	Cromosoma 2
Padres	(1.29 5.68 2.86 4.11 5.55)	(1.32 5.24 2.75 4.22 5.55)
Hijos	(1.29 5.68 2.75 4.22 5.55)	(1.32 5.24 2.86 4.11 5.55)

Ejemplo de crossover en n puntos de la representación por valores

Cuadro 2.11

2.6.3.3. Crossover uniforme

En este tipo de Crossover, cada posición tiene igual probabilidad de ser seleccionada de uno u otro cromosoma, es un crossover muy similar al de utilizar la denominada máscara de crossover.

	Cromosoma 1	Cromosoma 2
Padres	(1.29 5.68 2.86 4.11 5.55)	(1.32 5.24 2.75 4.22 5.55)
Hijos	(1.29 5.68 2.75 4.22 5.55)	(1.32 5.24 2.86 4.11 5.55)

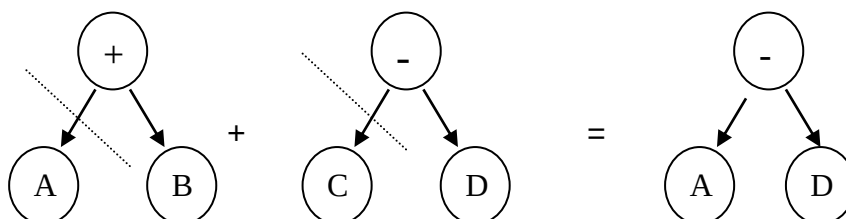
Ejemplo del crossover uniforme en la representación por valores

Cuadro 2.12

2.6.4. Crossover de la codificación mediante árboles

Se realiza un corte en alguna arista de los árboles, y se toma la parte izquierda de un padre, y la derecha del otro.

Por ejemplo:



Ejemplo de crossover en la representación mediante arboles

Figura 2.4

2.7. Mutación

En la Evolución, una mutación es un suceso poco común, como ya se ha visto anteriormente. En la mayoría de los casos las mutaciones son letales, pero en promedio, contribuyen a la diversidad genética de la especie. En un algoritmo genético tendrán el mismo papel, y la misma frecuencia (es decir, muy baja).

Una vez establecida la frecuencia de mutación, por ejemplo, uno por mil, se examina cada bit de cada cadena cuando se vaya a crear la nueva criatura a partir de sus padres (normalmente se hace de forma simultánea al crossover).

La gran ventaja real de la mutación es que no permitirá estancarse al algoritmo genético en una solución local, sino que permitirá dar saltos hacia otros puntos de la función, permitiendo de este modo una búsqueda global.

Además, no hace falta decir que no conviene abusar de la mutación. Es cierto que es un mecanismo generador de diversidad, y, por tanto, la solución cuando un algoritmo genético está estancado; pero también es cierto que reduce el algoritmo genético a una búsqueda aleatoria, lo cual es bastante lento.

Siempre es más conveniente usar otros mecanismos de generación de diversidad, como aumentar el tamaño de la población, o garantizar la aleatoriedad de la población inicial.

2.7.1. Mutación en codificación binaria.

Los tipos más comunes de mutación en este caso son:

- Mutación en un punto.
- Mutación en n puntos.
- Mutación uniforme.

2.7.1.1. Mutación en n puntos.

Debe tenerse claro el número de puntos a mutar, y si siempre mutará o no para cada punto. La mutación transformará el bit de 0 a 1 o de 1 a 0 según el caso.

Cromosoma antes de mutar	Cromosoma después de mutar
11001001	➔ 10011001

Ejemplo de mutación en n puntos de la representación binaria

Cuadro 2.13

2.7.1.2. Mutación Uniforme

Cada bit tendrá una diferente probabilidad para mutar; pero en todos los casos se comparará con la misma que se ingresó.

Cromosoma antes de mutar	Cromosoma después de mutar
11001001	→ 10101000

Ejemplo de mutación uniforme de la representación binaria

Cuadro 2.14

2.7.2. Mutación en codificación por permutaciones.

La más habitual es la de cambio de orden de las posiciones.

(1 2 3 4 5 6 8 9 7) => (1 8 3 4 5 6 2 9 7)
--

Ejemplo de mutación en la representación por permutaciones

Cuadro 2.15

2.7.3. Mutación en codificación por valores.

Una forma muy utilizada es la adición y substracción de una pequeña cantidad real a dos posiciones seleccionadas.

(1.29 5.68 2.86 4.11 5.55) => (1.29 5.68 2.75 4.22 5.55)
--

Ejemplo de mutación en la representación por valores

Cuadro 2.16

2.7.4. Mutación en codificación mediante árboles

Una forma de realizar la mutación en árboles es cambiando el operador o un número de éste.

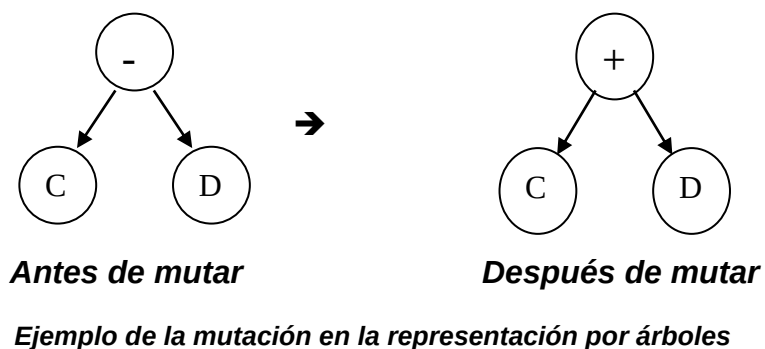


Figura 2.5

2.8. Otros operadores

No se usan en todos los problemas, sino sólo en algunos, y en principio su variedad es infinita. Generalmente son operadores que exploran el espacio de soluciones de una forma más ordenada, y que actúan más en las últimas fases de la búsqueda, en la cual se pasa de soluciones "casi buenas" a "buenas" soluciones.

2.8.1. Cromosomas de longitud variable

Hasta ahora se han descrito cromosomas de longitud fija, donde se conoce de antemano el número de parámetros de un problema; pero hay problemas en los que esto no sucede. Por ejemplo, en un

problema de clasificación, donde dado un vector de entrada, queremos agruparlo en una serie de clases, puede ser que no sepamos siquiera cuantas clases existen.

En diseño de redes neuronales, puede que no se sepa (de hecho, nunca se sabe) cuántas neuronas se van a necesitar. Por ejemplo, en un perceptrón hay reglas que dicen cuantas neuronas se deben de utilizar en la capa oculta; pero en un problema determinado puede que no haya ninguna regla heurística aplicable; tendremos que utilizar los algoritmos genéticos para hallar el número óptimo de neuronas.

En estos casos, necesitamos dos operadores más: *añadir* y *eliminar*. Estos operadores se utilizan para añadir un gen, o eliminar un gen del cromosoma. La forma más habitual de añadir un locus es *duplicar* uno ya existente, el cual sufre mutación y se añade al lado del anterior. En este caso, los operadores del algoritmo genético simple (selección, mutación, crossover) funcionarán de la forma habitual, salvo, claro está, que sólo se haga el crossover en la zona del cromosoma de menor longitud.

Estos operadores permiten, además, crear un *algoritmo genético de dos niveles*:

➤ Al nivel de cromosomas.

➤ Al nivel de genes.

Supongamos que, en un problema de clasificación, hay un gen por clase. Se puede asignar una puntuación a cada gen en función del número de muestras que haya clasificado correctamente. Al aplicar estos operadores, se duplicarán los aleles con mayor puntuación, y se eliminarán aquellos que hayan obtenido menor puntuación, o cuya puntuación sea nula.

2.8.2. Operadores de nicho (ecológico).

Otros operadores importantes son los operadores de *nicho*. Estos operadores están encaminados a mantener la diversidad genética de la población, de forma que cromosomas similares sustituyan sólo a cromosomas similares, y son especialmente útiles en problemas con muchas soluciones; un algoritmo genético con estos operadores es capaz de hallar todos los máximos de manera global y local, dedicándose cada especie a un máximo. Más que operadores genéticos, son formas de enfocar la selección y la evaluación de la población.

Los operadores de nicho más utilizados son:

- **Crowding o apiñamiento.**

Este operador trata de mantener la diversidad genética de la población, combinándose solamente entre cromosomas similares.

- **Discriminación o sharing.**

Consiste en introducir una función de discriminación que indica cuán similar es un cromosoma al resto de la población. La puntuación de cada individuo se dividirá por esta función de discriminación, de forma que se facilita la diversidad genética y la aparición de individuos diferentes.

2.8.3. Operadores especializados

En una serie de problemas hay que restringir las nuevas soluciones generadas por los operadores genéticos, pues no todas las soluciones generadas van a ser válidas, sobre todo en los problemas con restricciones. Por ello, se aplican operadores que mantengan la estructura del problema. Otros operadores son simplemente generadores de diversidad, pero la generan de una forma determinada.

Los operadores especializados más conocidos son:

- **Zap**

En lugar de cambiar un solo bit de un cromosoma, lo cambia completamente.

- **Creep**

Este operador aumenta o disminuye en 1 el valor de un gen; sirve para cambiar suavemente y de forma controlada los valores de los genes.

- **Transposición**

Es similar al crossover y a la recombinación genética, pero dentro de un solo cromosoma. Dos genes intercambian sus valores, sin afectar al resto del cromosoma. Similar a este es el operador de eliminación-reinserción, en el que un gen cambia de posición con respecto a los demás.

2.9. La población

2.9.1. Técnicas basadas en población

Este tipo de técnicas pueden ser versiones de cualquiera de las anteriores, pero en vez de tener una sola solución, que se va alterando hasta obtener el óptimo, se persigue el óptimo cambiando varias soluciones; de esta forma es más fácil escapar

de los mínimos locales tan temidos. Entre estas técnicas se hallan la mayoría de los algoritmos evolutivos.

2.9.2. Tamaño de la población

Una cuestión que uno puede plantearse es la relacionada con el tamaño idóneo de la población. Parece intuitivo que las poblaciones pequeñas corren el riesgo de no cubrir adecuadamente el espacio de búsqueda, mientras que el trabajar con poblaciones de gran tamaño puede acarrear problemas relacionados con el excesivo costo computacional.

Goldberg efectuó un estudio teórico, obteniendo como conclusión que el tamaño óptimo de la población para ristas de longitud l , con codificación binaria, crece exponencialmente con el tamaño de la rista.

Este resultado traería como consecuencia que la aplicabilidad de los algoritmos genéticos en problemas reales sería muy limitada, ya que resultarían no competitivos con otros métodos de optimización combinatoria. Alander, basándose en evidencia empírica sugiere que un tamaño de población comprendida entre 1 y 21 es suficiente para atacar con éxito los problemas considerados por él.

2.9.3. La población inicial

Habitualmente la población inicial se escoge generando ristas al azar, pudiendo contener cada gen a uno de los posibles valores del alfabeto con probabilidad uniforme. Nos podríamos preguntar que es lo que sucedería si los individuos de la población inicial se obtuviesen como resultado de alguna técnica heurística o de optimización local. En los pocos trabajos que existen sobre este aspecto, se constata que esta inicialización no aleatoria de la población inicial, puede acelerar la convergencia del algoritmo genético. Sin embargo en algunos casos la desventaja resulta ser la prematura convergencia del algoritmo, queriendo indicar con esto la convergencia hacia óptimos locales.

2.10. Otras técnicas de programación evolutiva

2.10.1. Redes neuronales

Considero importante comparar el funcionamiento de los algoritmos genéticos con otras técnicas informáticas robustas en optimización, por lo que en esta etapa se explica un poco sobre las redes neuronales.

2.10.1.1. La neurona

A finales del siglo 19 se logró una mayor claridad sobre el trabajo del cerebro debido a los trabajos de Ramón y Cajal en España y Sherrington en Inglaterra. El primero trabajó en la anatomía de las neuronas y el segundo en los puntos de conexión de las mismas o sinápsis.

El tejido nervioso es el más diferenciado del organismo y está constituido por células nerviosas, fibras nerviosas y la neuroglia, que está formada por varias clases de células. La célula nerviosa se denomina neurona, que es la unidad funcional del sistema nervioso. Hay neuronas bipolares, con dos prolongaciones de fibras y multipolares, con numerosas prolongaciones. Estas pueden ser neuronas sensoriales, motoras y de asociación.

Se estima que en cada milímetro del cerebro hay cerca de 50.000 neuronas. El tamaño y la forma de las neuronas es variable, pero con las mismas subdivisiones que muestra la figura. El cuerpo de la neurona o Soma contiene el núcleo. Se encarga de todas las actividades metabólicas de la neurona y recibe la información de otras neuronas vecinas a través de las conexiones sinápticas.

Las dendritas son las conexiones de entrada de la neurona y por su parte el axón es la "salida" de la neurona y se utiliza para enviar impulsos o señales a otras células nerviosas. Cuando el axón esta cerca de sus células destino se divide en muchas ramificaciones que forman sinápsis con el soma o axones de otras células. Esta unión puede ser inhibidora o excitadora según el transmisor que las libere. Cada neurona recibe de 10.000 a 100.000 sinápsis y el axón realiza una cantidad de conexiones similar.

La transmisión de una señal de una célula a otra por medio de la sinápsis es un proceso químico. En él se liberan sustancias transmisoras en el lado del emisor de la unión. El efecto es elevar o disminuir el potencial eléctrico dentro del cuerpo de la célula receptora. Si su potencial alcanza el umbral se envía un pulso o potencial de acción por el axón. Se dice, entonces, que la célula se disparó. Este pulso alcanza otras neuronas a través de las distribuciones de los axones.

2.10.1.2. La red neuronal

El sistema de neuronas biológico esta compuesto por neuronas de entrada (censores) conectados a una compleja red de neuronas calculadoras (neuronas ocultas), las cuales a su vez están

conectadas a las neuronas de salidas que controlan, por ejemplo, los músculos.

Los sensores pueden ser señales emitidas por los oídos, los ojos, etc. Las respuestas de las neuronas de salida activan los músculos correspondientes. En el cerebro hay una gigantesca red de neuronas calculadoras u ocultas que realizan la computación necesaria. De esta manera similar, una red neuronal artificial debe ser compuesta por sensores del tipo mecánico o eléctrico.

2.10.1.3. Las redes neuronales artificiales

2.10.1.3.1. Sistema experto.

Un método más avanzado para representar el conocimiento, es el sistema experto. Típicamente está compuesto por varias clases de información almacenada: Las reglas Si - Entonces le dicen al sistema como se debe reaccionar ante los estados del mundo. Una regla del sistema experto puede ser Si Y es un hombre, Entonces Y es mortal. Los hechos describen el estado del mundo. Por ejemplo: Juan es mortal. Por último, una máquina de inferencia relaciona los hechos conocidos con las reglas Si - Entonces y genera una conclusión. En el ejemplo: Juan es mortal, esta nueva conclusión se añade a la colección de hechos que se

almacena en los medios ópticos o magnéticos del computador digital. De esta forma, un sistema experto sintetiza nuevo conocimiento a partir de su entendimiento del mundo que le rodea, y de esta forma, un sistema experto es un método de representación y procesamiento del conocimiento mucho más rico y poderoso que un simple programa de computador. Sin embargo, con respecto a la manera en que opera el cerebro humano, las limitaciones son múltiples. Los problemas planteados en términos difusos o ambiguos, por ejemplo, son muy complejos de analizar o de conocer con sistemas de procesamiento simbólico, tales como los sistemas expertos o programas de computador.

2.10.1.3.2. Interpretación de la neurona por computadora.

Una neurona se puede comparar con una caja negra compuesta por varias entradas y una salida. La relación de activación entre la salida y la entrada, o en términos circuitales o de teoría de control.

La variable **f** es la frecuencia de activación o emisión de potenciales y **u** es la intensidad del estímulo de la soma.

2.10.1.3.2.1. La neurona artificial

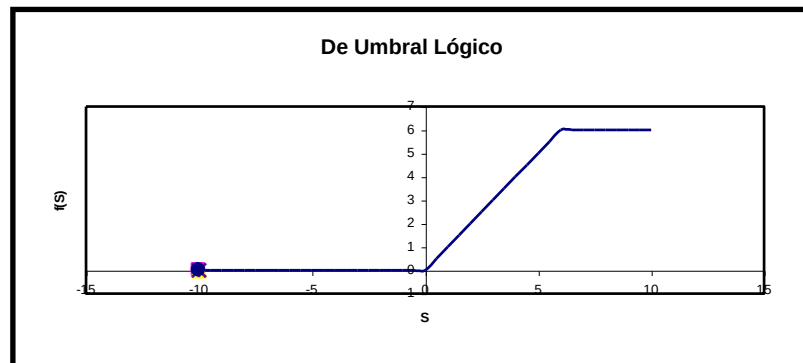
Es un circuito eléctrico que realice la suma ponderada de las diferentes señales que recibe de otras unidades iguales y

produzca en la salida un **uno** o un **cero** según el resultado de la suma con relación al umbral o nivel de disparo, conforma una buena representación de lo que es una neurona artificial. La función de transferencia para la activación o disparo de la neurona puede ser:

- * de umbral lógico
- * de limitación dura, o
- * de función tipo S.

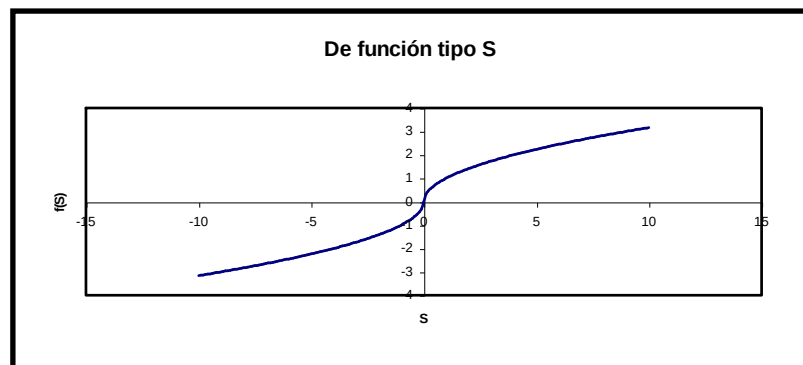
W representa el peso o ponderación de la conexión a través de una entrada.

La neurona artificial es un dispositivo eléctrico que responde a señales eléctricas. La respuesta la produce el circuito activo o función de transferencia que forma parte del cuerpo de la neurona. Las dendritas llevan las señales eléctricas al cuerpo de la misma. Estas señales provienen de sensores o son salidas de neuronas vecinas. Las señales por las dendritas pueden ser voltajes positivos o negativos; los voltajes positivos contribuyen a la excitación del cuerpo y los voltajes negativos contribuyen a inhibir la respuesta de la neurona.



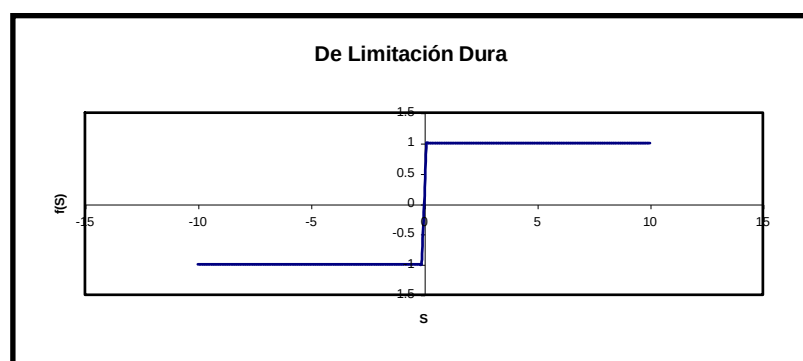
Función de transferencia de las redes neuronales de tipo umbral lógico

Figura 2.6



Ejemplo de una función de transferencia de las redes neuronales de tipo S

Figura 2.7



Ejemplo de una función de transferencia de las redes neuronales de tipo limitación dura

Figura 2.8

2.10.1.3.2.2. Entrenamiento

Cuando el sistema humano de neuronas, los ojos captan un objeto A, por ejemplo, algunos de los sensores de la visión se activan y envían señales a las neuronas ocultas. Las neuronas que se disparan con la señal de entrada aumentan el grado de conexión de ellas. Si el mismo objeto A se presenta una y otra vez, la interconexión de neuronas se refuerza y, por lo tanto, el conocimiento del objeto.

Si se le presenta a la red anterior el objeto A modificado, la unión de las neuronas para el conocimiento de tal objeto, es débil. Las neuronas deben entrenarse para reconocer el objeto A en esta nueva presentación. Luego de algunas sesiones de entrenamiento, el sistema neuronal es capaz de reconocer el objeto A en todas sus formas. Si el objeto cambia nuevamente la red de neuronas y el conocimiento se actualizan.

Este entrenamiento, repetido para todos los valores de entrada y salida que se quiera, origina una representación interna del objeto en la red, que considera todas las irregularidades y generalidades del mismo.

2.10.1.4. Método de transmisión de la información en el cerebro

Antes conviene saber que en los primeros tiempos de la informática a los ordenadores se los llamaba calculadoras de cifras electrónicas o simplemente calculadoras digitales. Los sistemas digitales trabajan con cifras en código binario que se transmiten en formas de impulsos (bits). Los sistemas analógicos procesan señales continuamente cambiantes, como música o palabra hablada.

Por suerte para nuestro propósito de imitar con un ordenador el cerebro este también codifica la información en impulsos digitales. En los humanos las sensaciones se generan digitalmente y se transmiten así a través del sistema nervioso. Con otras palabras cuando la luz se hace más intensa, el sonido mas alto o la presión mas fuerte, entonces no es que fluya mas corriente a través de los nervios, sino que la frecuencia de los impulsos digitales aumenta.

En principio los ordenadores trabajan de manera semejante. Así una sensación mas fuerte corresponde en un equipo informático a una cifra más alta (o en una palabra mas larga). Sin embargo en un ordenador los datos se transmiten siempre a un mismo ritmo; la frecuencia base es inalterable. Por eso las cifras mas altas tardan

mas tiempo en ser transmitidas. Como por lo general el ordenador o trabajan en tiempo real, esto no tiene mayor importancia, pero cuando se trata de un procesador en tiempo real, como son los empleados en proceso industrial, hace falta de ampliar el numero de canales de transmisión para que en el mismo espacio de tiempo pueda fluir mayor cantidad de datos.

2.10.1.4.1. Compuertas lógicas

Sabemos que los elementos básicos de un ordenador son las compuertas lógicas, en el cerebro también existen aunque no son idénticas a las de un ordenador.

En un ordenador las compuertas de conjunción, disyunción, etc. tienen una función perfectamente determinada e inalterable. En el cerebro también hay elementos de conexión parecidos, las llamadas sinapsis, donde confluyen en gran numero las fibras nerviosas.

2.10.1.4.1.1. Funcionamiento de las sinapsis

Cientos de datos fluyen por los nervios hasta cada sinapsis, donde son procesados. Una vez analizada y tratada la información esta sale ya transformada por los canales nerviosos.

En los seres vivos no pueden permitirse el lujo de la especialización ya que si algo se rompe otro elemento debe hacerse cargo de la función. Por eso cada sinapsis es simultáneamente una compuerta de conjunción, disyunción, negación, etc.

Una sinapsis suma las tensiones de los impulsos entrantes. Cuando se sobrepasa un determinado nivel de tensión; el llamado umbral de indicación; esta se enciende, esto es deja libre el camino para que pasen los impulsos. Si el umbral de indicación de tensión es demasiado bajo, la sinapsis actúa como una puerta lógica del tipo disyuntivo, pues en tal caso pocos impulsos bastan para que tenga lugar la conexión. En cambio cuando el umbral de indicación es alto, la sinapsis actúa como una puerta de conjunción, ya que en ese caso hace falta que lleguen la totalidad de los impulsos para que el camino quede libre. También existen conducciones nerviosas que tienen la particularidad de bloquear el paso apenas llegan los impulsos. Entonces la sinapsis hace la función de una compuerta inversora. Esto demuestra la flexible del sistema nervioso.

2.10.1.5. Diferencias entre el cerebro y un ordenador

La diferencia más importante y decisiva es el cómo se produce el almacenamiento de información en el cerebro y en el ordenador.

Ordenador: Los datos se guardan en posiciones de memoria que son celdillas aisladas entre sí. Así cuando se quiere acceder a una posición de memoria se obtiene el dato de esta celdilla. Sin que las posiciones de memoria adyacentes se den por aludidas.

- * Sistemas altamente especializados con capacidad para procesar información muy concreta, siguiendo unas instrucciones dadas.
- * La frecuencia de transmisión es inalterable y esta dada por el reloj interno de la máquina.
- * Las compuertas lógicas tienen una función perfectamente determinada e inalterable.
- * La información se guarda en posiciones de memoria de acceso directo por su dirección.
- * En el interior de una computadora los impulsos fluyen a la velocidad de la luz.

Cerebro: La gestión es totalmente diferente. Cuando buscamos una información no hace falta que sepamos donde se encuentra

almacenada y en realidad no lo podemos saber ya que nadie sabe donde guarda hasta hoy en día el cerebro los datos.

- * Sistema de datos de múltiple propósito capaz de tratar gran cantidad de información en poco tiempo pero no necesariamente con exactitud.
- * La frecuencia de los impulsos nerviosos puede variar.
- * Las llamadas sinapsis cumple en el cerebro la función simultánea de varias compuertas (and, or, not etc.)
- * La memoria es del tipo asociativo y no se sabe dónde quedara almacenada.
- * Los impulsos fluyen a 30 metros por segundo.

Pero tampoco es necesario ya que basta con que pensemos en el contenido o significado de la información para que un mecanismo, cuyo funcionamiento nadie conoce, no nos proporcione automáticamente solo la información deseada sino que también las informaciones vecinas, es decir, datos que de una u otra manera hacen referencia a lo buscado.

Los expertos han concebido una serie de tecnicismos para que lo incomprensible resulte algo más comprensible. Así a nuestro sistema para almacenar información se lo llama memoria asociativa. Esta expresión quiere dar a entender que los humanos

no memorizan los datos direccionándolos en celdillas, sino por asociación de ideas; esto es, interrelacionando contenidos, significados, modelos.

En todo el mundo pero sobre todo en Estados Unidos y Japón, científicos expertos tratan de dar con la clave de la memoria asociativa. Si se consiguiera construir un chip de memoria según el modelo humano, la ciencia daría un paso gigante en la fascinante carrera hacia la inteligencia artificial. Y además el bagaje del saber humano quedaría automáticamente enriquecido.

2.10.1.6. Similitudes entre el cerebro y una computadora

- ⌘ Ambos codifican la información en impulsos digitales.
- ⌘ Tanto el cerebro como la computadora tienen compuertas lógicas.
- ⌘ Existen distintos tipos de memoria.
- ⌘ Los dos tienen aproximadamente el mismo consumo de energía.

2.10.1.7. Un superordenador llamado cerebro

El hombre necesita un sistema de proceso de datos de múltiple propósito capaz de tratar gran cantidad de información muy distinta y en muy poco tiempo y con el mayor sentido practico(pero

no necesariamente con exactitud), para inmediatamente poder actuar en consecuencia. Los ordenadores, en cambio, son altamente especializados con capacidad para procesar con exactitud información muy concreta(en principio solo números) siguiendo unas instrucciones dadas.

El cerebro humano posee mas de diez millones de neuronas las cuales ya están presentes en el momento del nacimiento conforme pasa el tiempo se vuelven inactivas, aunque pueden morir masivamente.

Nuestro órgano de pensamiento consume 20 Patios/hora de energía bioquímica, lo que corresponde a una cucharada de azúcar por hora. Los ordenadores domésticos consumen una cantidad semejante. Las necesidades de oxígeno y alimento es enorme en comparación con el resto del cuerpo humano: casi una quinta parte de toda la sangre fluye por el cerebro para aprovisionar de oxígeno y nutrientes. La capacidad total de memoria es difícil de cuantificar, pero se calcula que ronda entre 10^{12} y 10^{14} bits.

La densidad de información de datos de un cerebro todavía no se ha podido superar artificialmente y en lo que se refiere a velocidad de transmisión de datos, a pesar de la lentitud con que transmite

cada impulso aislado, tampoco esta en desventaja, gracias a su sistema de proceso en paralelo: la información recogida por un ojo representa 10^6 bits por segundo.

Según todos los indicios el cerebro dispone de dos mecanismos de almacenamiento de datos: la memoria intermedia acepta de cinco a diez unidades de información, aunque solo las mantiene durante algunos minutos. La memoria definitiva guarda las informaciones para toda la vida, lo que no significa que nos podamos acordar siempre de todo. La memoria inmediata trabaja como una espacie de cinta continua: la información circula rotativamente en forma de impulsos eléctricos por los registros. El sistema es comparable a la memoria dinámica de un ordenador, en la que la información tiene que ser refrescada continuamente para que no se pierda. En cambio, la memoria definitiva parece asemejare mas bien a las conocidas memorias de celdillas de los ordenadores. Se cree que esta memoria funciona gracias a formaciones químicas de las proteínas presentes en el cerebro humano.

2.10.1.8. Aplicaciones de las redes neuronales

Las redes neuronales son una gran herramienta de optimización, porque aplican el conocimiento que acumulan por medio de premisas, por lo que se han desarrollado diversas aplicaciones tales como:

- ☒ Optimización de rutas
- ☒ Aplicaciones de inferencias lógicas
- ☒ Sistemas de toma de decisiones
- ☒ Juegos de tablero

De igual manera se ha llegado a simular partes del cuerpo humano que actualmente se utilizan, y otras que se utilizarán en el futuro, así como:

- ☒ Retina artificial
- ☒ Oído artificial
- ☒ Motricidad asistida
- ☒ Minibomba para diabéticos
- ☒ Minidesfibrilador
- ☒ Corazón Artificial
- ☒ Descargas eléctricas contra la epilepsia

2.10.1.9. Redes neuronales y algoritmos genéticos

Las redes neuronales y los algoritmos genéticos se utilizan para extraer automáticamente conocimiento útil para el proceso de modelaje y previsión.

Las redes neuronales son una potente herramienta de modelaje:

- Aprenden los patrones de un conjunto de datos de entrenamiento
- Poseen un buen comportamiento en presencia de ruido
- Pueden modelar relaciones complejas
- Son capaces de generalizar para predecir

Los algoritmos genéticos se utilizan para buscar el mejor modelo cuando la cantidad de combinaciones de modelos posibles es demasiado grande. Utilizan una función de evaluación para determinar la calidad de los individuos de la población.

En nuestro caso, para la selección del mejor modelo de forma automática se utiliza como criterio un estadístico que tiene en cuenta la calidad de los residuos y la cantidad de parámetros.

2.11. Autómatas Celulares

Los autómatas celulares son redes de autómatas simples conectados localmente. Cada autómatas simple produce una

salida a partir de varias entradas, modificando en el proceso su estado según una función de transición. Por lo general, en un autómata celular, el estado de una célula en una generación determinada depende única y exclusivamente de los estados de las células vecinas y de su propio estado en la generación anterior.

Los autómatas celulares son herramientas útiles para modelar cualquier sistema en el universo. Pueden considerarse como una buena alternativa a las ecuaciones diferenciales y han sido utilizadas para modelar sistemas físicos, como interacciones entre partículas, formación de galaxias, cinética de sistemas moleculares y crecimiento de cristales, así como diversos sistemas biológicos a nivel celular, multicelular y poblacional.

La Vida Artificial es un mecanismo para la representación de sistemas complejos. Una vez representado un sistema, podemos experimentar con él en situaciones hipotéticas.

En toda ciencia existe simplificación o generalización; esto es lo que nos permite razonar, extraer conclusiones o crear hipótesis sobre nuevos problemas gracias a los ejemplos del pasado. Una simulación nunca es perfecta, lo contrario sería realizar una copia idéntica, átomo por átomo (átomo como unidad indivisible, si

existiera tal concepto). Se trata de no tener que construir una central nuclear para experimentar y prever comportamientos en una central nuclear. Los sistemas complejos más completos de todos serían aquellos cuya única representación aceptable serían ellos mismos.

En los sistemas complejos puede producirse lo que se denomina el comportamiento emergente. La gran ventaja de la Vida Artificial es que permite observar fenómenos emergentes. Las propiedades emergentes son aquellas que aparecen en los sistemas como resultado de la interacción entre sus partes y que no pueden explicarse a partir de las propiedades de los elementos que lo componen.

Toda ciencia tiende al reduccionismo, a explicar fenómenos complejos por su descomposición en otros más simples. Sin embargo existen ciencias como la ecología (que se dedica al estudio de cómo los organismos, estudiados individualmente por otras ramas de la biología, se insertan en el mundo real) no puede dejar de observarse el sistema en conjunto, no puede olvidarse el punto de vista hollístico, es decir, la convergencia de comportamiento que resulta cuando diversos elementos del

sistema se condicionan mutuamente. No es posible, o no basta, estudiar los elementos por separado.

En estos sistemas, una acción simple se puede propagar exponencialmente provocando efectos de gran envergadura. Por esta razón se dice que estos sistemas se encuentran en la frontera entre el orden y el desorden.

Por ejemplo, un montón de arena sobre el que dejamos caer granos lentamente. Ocurre que a partir de cierto ángulo la caída de un sólo grano provoca avalanchas de gran tamaño. De la misma forma, la caída de un copo de nieve puede provocar la ruptura de la rama de un árbol sobre la que se ha acumulado gran cantidad de copos.

Una colonia de hormigas naturales puede llevar a cabo tareas de gran complejidad como explorar el entorno, construir galerías, o decidir si es adecuado un tipo de alimento, con la lamentable pérdida de algunas hormigas en el proceso. Pero las hormigas, consideradas una por una, son incapaces de realizar ninguna de estas tareas.

El comportamiento del hormiguero emerge a partir de interacciones entre elementos simples (hormigas) y no se puede

explicar únicamente basándonos en las propiedades de un individuo. El hormiguero podría considerarse como un ser vivo de un nivel superior a la hormiga.

2.11.1. Comparación entre los algoritmos genéticos y los autómatas celulares

En un algoritmo genético es difícil saber cuándo una estructura de datos es adecuada; sin embargo, sí existe una forma de saber por anticipado cuándo una estructura de datos no es adecuada.

Para que el algoritmo funcione como deseamos, la reproducción y las mutaciones deberían producirse de tal forma que mantengan, en la mayoría de los casos, las cualidades positivas de los individuos. Pensemos en el caso de las mutaciones. Dada una cadena, una mutación consistirá en modificar al azar uno de los elementos de esa cadena. Para que todo marche bien deberá ocurrir que los valores de los genes sean buenos o malos, independientemente de los valores de otros genes, o al menos, que sean lo más independientes posible.

Un buen método para saber cuándo una estructura de datos no es adecuada, es el siguiente: si al modificar un elemento cualquiera de una cadena seleccionada (buena) existe una probabilidad alta

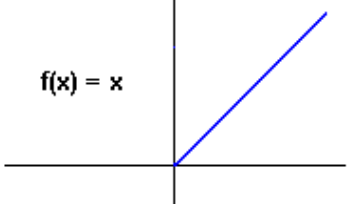
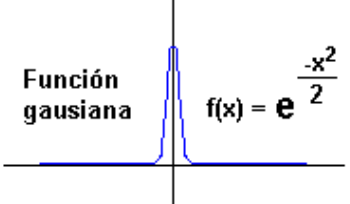
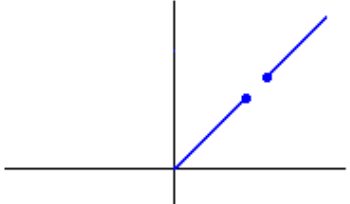
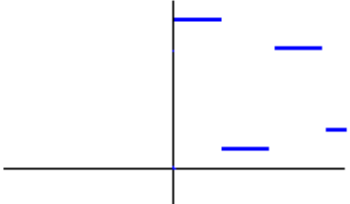
de que la cadena resultante sea pésima, entonces la estructura de datos elegida no es correcta, ya que los componentes de la cadena por sí solos no contienen la información que hace que esa cadena sea seleccionada.

Es decir, *"Cuando el objetivo (peso) de una entidad se puede calcular como una función en la que variaciones muy pequeñas de los valores de sus variables (valores de los genes) producen con una probabilidad alta, variaciones muy grandes en el resultado de la función, la estructura de datos es incorrecta"*. Una variación muy pequeña sería cambiar el valor de un gen.

El ejemplo por excelencia de este tipo de problemas es la lotería. El cambio de uno sólo de los dígitos de nuestro billete puede producir cambios enormes en la cantidad que se recibe como premio. Un algoritmo genético que trate de predecir el número premiado de esta forma nunca tendrá demasiado éxito.

La descripción que se acaba de dar de los problemas que hay que evitar suena mucho a funciones discontinuas; sin embargo, se habla de estos problemas como funciones no lineales. ¿Qué es lo que ocurre aquí? Vamos a verlo con detenimiento. Una función puede ser continua o discontinua, lineal o no lineal. En la siguiente figura se ilustran los cuatro casos.

La confusión de la que se está tratando se hizo patente en el artículo "Algoritmos Genéticos Avanzados" publicado en la revista Solo Programadores (Nº 37 de Septiembre de 1997) donde se identificaron erróneamente las funciones no lineales con aquellas en la que variaciones pequeñas en las variables producen variaciones grandes en el valor de la función. La afirmación era errónea. Lo que llevó al error es que ambas definiciones hacen referencia al mismo problema en el caso de los algoritmos genéticos.

Funciones	LINEAL	NO LINEAL
CONTINUA	$f(x) = x$ 	Función gaussiana $f(x) = e^{-\frac{x^2}{2}}$ 
DISCONTINUA		

Tipos de funciones en cuanto a linealidad y continuidad

Figura 2.12

Al ser los valores de los genes discretos, el valor de la función también lo será, y podemos representar los posibles valores del

peso como una nube de puntos. Dado que la función sólo puede saltar de un valor a otro, podemos identificar dos tipos de funciones: aquellas en las que al realizar un pequeño cambio en sus variables, se producen siempre saltos pequeños (en cuanto a peso) o aquellas en las que es probable que el salto sea muy grande, correspondiendo el segundo caso como un problema que de difícil solución.

En las gráficas de ejemplo, los tres primeros casos corresponden con problemas que pueden ser resueltos por algoritmos genéticos. El último de ellos (función discontinua y no lineal) corresponde con un problema de difícil solución, ya que no es posible acercarse gradualmente al objetivo buscado, o lo que es lo mismo, no hay una presión selectiva que dirija a los individuos hacia el máximo.

Los problemas en los que encontramos este tipo de inconvenientes han sido identificados por John H. Holland (el "padre" de los algoritmos genéticos) como problemas no lineales, y ciertamente son así. En el artículo "Algoritmos Genéticos" publicado en la Revista Investigación y Ciencia, en el número de Septiembre 1992. En las páginas 41 y 42 aparece:

"El paralelismo implícito del algoritmo genético le permite ensayar un gran número de regiones del espacio de búsqueda

manipulando un número relativamente reducido de ristas. También le facilita afrontar problemas no lineales, problemas en las que la adecuación de una cadena que contenga dos bloques constructivos concretos puede ser mucho mayor (o mucho menor) que la suma de adecuaciones atribuibles por separado a cada bloque constructivo individual".

Es decir, que si ocurre que un gen es bueno o malo en función del valor de otros genes, el problema se nos complica. ¿Por qué no habla Holland de funciones discontinuas y además no lineales?. La función gaussiana es no lineal y sin embargo sí puede ser resuelta con un algoritmo genético. La explicación a esto se encuentra en lo que ya se ha mencionado. Como se trata de variables discretas, la función siempre será discontinua, por lo que estrictamente podemos ignorar las funciones continuas, y referirnos a este tipo de problemas simplemente como funciones no lineales.

Una vez aclarado esto, se van a puntualizar otras cosas que pueden no quedar claras. Aunque se están haciendo hipótesis sobre una determinada función, dicha función es desconocida, precisamente se trata de hallar el máximo (o varios máximos) sin saber cómo es realmente la función. Si la conociésemos, bastaría

derivarla e igualar a cero para hallar el máximo, es decir, la mejor entidad. Solo sabemos que:

$$\text{peso} = f(\text{valor_de_gen_1}, \text{valor_de_gen_2}, \dots)$$

Curiosamente, lo que sí somos capaces es de asignar un peso a un individuo o conjunto de valores de genes. Es decir, sabemos reconocer en que grado se cumple aquello que buscamos, aunque no sabemos cómo encontrarlo.

Esto es evidente y no hay otra opción. Si no podemos reconocer aquello que buscamos, es imposible encontrarlo. Cuanta más exactitud tengamos en identificar el grado de bondad de cada solución, tanto mejor funcionará el algoritmo.

Volviendo a la no-linealidad, en nuestros problemas ojalá siempre ocurriera que:

$$\text{objetivo} = f1(\text{valor_de_gen_1}) + f2(\text{valor_de_gen_1}) + f3(\text{valor_de_gen_1})$$

O lo que es lo mismo: ciertos valores para ciertos genes son buenos, y otros son malos, pero la bondad o maldad de cada uno es independiente de los valores de los otros.

Por ejemplo, tenemos la función $Y = 3X + 4Z + 17$

En esta función variaciones pequeñas de X ó Z producen variaciones pequeñas de Y.

En cambio en la función

$$Y = \begin{cases} 3X & \text{para } Z < > 1 \\ 400 & \text{para } Z = 1 \end{cases}$$

Ocorre que, por ejemplo, Y tiene el valor 400 si Z=1, pero para Z=1.0000000000001 tenemos que Y = 3 * X, que no tiene porqué estar cercano al valor 400.

En un algoritmo genético, si el patrón posee 4 elementos A, B, C y D cuyos valores pueden oscilar entre 0 y 10, una función objetivo adecuada podría ser:

$$\text{objetivo} = (2 * A) + (4 * B) - (2 * C)$$

En este caso se observa que valores altos de A producen siempre una gran proximidad al objetivo, lo mismo ocurre con los valores de B, pero con un efecto del doble que A. En cambio, los valores de C deberán ser lo más pequeños posibles, ya que disminuyen el objetivo, siendo indiferentes los de D.

Así, por ejemplo, al reproducir una solución, como por ejemplo:

A = 5; B = 4; C = 1; D = 9 contra otra, es muy probable que las cualidades positivas (valores altos de A y B; valores bajos de C) se transmitan a la descendencia.

El tres en raya con reglas es bastante lineal: cada regla, o es buena, o es mala. Si una entidad seleccionada posee 40 o 50 reglas, probablemente podamos cambiar una sin que se convierta en un pésimo jugador.

Pero en casi todos los problemas existe una cierta no-linealidad inevitable. Por ejemplo, pueden producirse por azar reglas muy interesantes, pero que no son seleccionadas porque necesitan de otras para demostrar su utilidad y producir un peso alto. Es decir, las no linealidades son trozos de cadena cuya utilidad depende de los valores de otros segmentos, y que sólo son útiles cuando todos ellos contienen determinados datos, desapareciendo la utilidad al fallar uno de ellos.

Cuanta más no-linealidad exista, más lentamente convergerá nuestro algoritmo genético. Esto es completamente inevitable, ya que no es posible encontrar algo si no sabemos qué es lo que estamos buscando. Sin embargo, la naturaleza parece que sí lo consigue. Así cómo han podido generarse gradualmente órganos complejos, tales como los ojos o las plumas para el vuelo, que no

dependen de una única mutación, si solamente el órgano completo es útil al individuo. No siempre parece fácil explicar todos los rasgos que presenta un organismo con un mecanismo adaptativo.

En mi opinión, este planteamiento es erróneo. Los seres vivos sacan beneficio de cualquier órgano a medio formar. Un pato puede huir de un depredador terrestre alzando el vuelo tan sólo un metro. El sistema visual de la rana está sintonizado solamente a cuatro clases de estímulo, estos son: el contraste de luz y oscuridad; un borde de luz u oscuridad; una repentina disminución de la luminosidad; y el constante movimiento de un pequeño objeto de color negro.

Es habitual considerar nuestra inteligencia o la visión de un halcón como perfectas, en vez de simples estadios intermedios, que llegarán a niveles mucho más avanzados gracias a la evolución. La naturaleza probablemente sea mucho más lineal de lo que parece, y nosotros podemos ser la rana.

Entonces, los algoritmos genéticos tal vez logren llegar a una solución óptima pero tardarán muchísimo en este tipo de funciones discontinuas y no lineales.

2.11.2. Aplicaciones de los autómatas celulares

Al mencionar los autómatas celulares, como se escribió anteriormente, se simularan y resolverán situaciones en conjunto, que como elementos independientes no se pueden evaluar conocimientos. Entonces puede mencionares como aplicaciones de los autómatas celulares:

- ✎ Inteligencia artificial basada en el comportamiento.
- ✎ Ecosistemas artificiales.
- ✎ Vida artificial en la construcción de un espacio cibernético.

2.12. Observaciones

Con este capítulo he dado una explicación completa de las diferentes formas de representar los algoritmos genéticos, y he explicado no solamente la parte que necesito en este trabajo, sino todo lo referente a los mismos, y además he realizado una comparación con las redes neuronales y con los autómatas celulares que son otras técnicas de programación evolutiva.

CAPITULO III

3. Modelo poblacional: variables poblacionales

3.1. Introducción

En este capítulo se explica el funcionamiento de los modelos poblacionales clásicos, así como se hace mención a los modelos poblacionales estocásticos y la interacción entre dos o más poblaciones, y finalmente se explicará cómo se aplicará el paradigma de los algoritmos genéticos a los modelos poblacionales, dando una propuesta que se presenta interesante de un modelo de crecimiento poblacional.

3.2. La población

Es el conjunto de individuos que pertenecen a una misma especie y deben estar coexistiendo y por lo tanto están en interacción unos

con otros, pudiendo cruzarse entre sí de forma tal que comparten una gran cantidad de información común. Se define como un conjunto de individuos que tienen la capacidad potencial de cruzarse dejando descendencia fértil. Lo importante es la capacidad potencial de reproducción.

Propiedades emergentes del nivel poblacional: Algunas de las propiedades del nivel poblacional son: densidad, disposición espacial, estructura de edades y sexo, modelo de crecimiento y parámetros poblacionales. Cada población de una misma especie, tendrá sus propios parámetros que lo caracterizan.

3.2.1. Interacción entre las poblaciones

Para que los organismos interactúen deben coexistir y presentar superposición de nicho, en alguna de sus dimensiones. La variedad de requerimiento y de estrategia crea una intrínseca gama de interrelaciones que implica beneficios y también de restricciones, por lo cual, generalmente, son mecanismos de regulación de la comunidad.

3.2.2. Los modelos poblacionales

La Humanidad, según Roberth Malthus, aumenta en progresión geométrica, en tanto que la producción crece en progresión

aritmética. La solución que propone para este supuesto desajuste es negar la asistencia a los pobres, para que así no tengan hijos. El modelo de Malthus es una interpretación elitista de la nueva situación que se presenta en la Europa de la Revolución industrial, durante el siglo XIX, con la proliferación del proletariado en las ciudades fabriles.

La afirmación "la población crece en forma geométrica mientras que la producción de alimentos lo hace en forma aritmética"; pareciera haber sido refutada al menos desde la visión del mundo tecnológico y económico. Las terribles hambrunas que estabilizaban la población en los siglos anteriores parecían ser la manera razonable en la que el mundo contenía el crecimiento. Sin embargo en la actualidad, dejando de lado el problema de la distribución de los alimentos, hasta ahora la tecnología acompaña cada uno de los estirones que pega la población mundial.

Sin embargo las estimaciones de las Naciones Unidas predicen un incremento de 1000 millones de personas cada 11 años, de acá al 2020. En ese año habrá en el mundo un tercio más de gente que ahora, otros 2000 millones sobre los casi 6000 actuales. ¿Podrá la tecnología crecer tan rápido como la población? ¿Y en ese caso, cuál será el costo ambiental?

La crisis económica de los años treinta, que se acompaña de un acusado descenso de la natalidad en el mundo occidental, permite formular el enunciado del modelo de la transición demográfica. Según este modelo, la evolución de la población sigue distintas fases. La primera corresponde a una etapa de altas tasas de natalidad y mortalidad, que producen un lento crecimiento de la población. Le sigue una etapa intermedia, de alta fecundidad y baja mortalidad, con un crecimiento muy acentuado en algún momento. Y el ciclo terminaría con bajas pautas, tanto de natalidad como de mortalidad, y un crecimiento lento o estable.

El modelo de la explosión demográfica o solución neomalthusiana, se enuncia por primera vez a fines de los años cuarenta, en 1947, con el naturalista norteamericano Vogt, y se basa en una nueva situación, apenas prevista en el modelo anterior. La población experimenta un elevado crecimiento que, según este modelo, se prolongaría en el futuro, superando con creces el desarrollo económico e incluso comprometiéndolo. De no cortarse este crecimiento, la Humanidad caminaría hacia su autodestrucción. En tanto que Malthus indica la necesidad de que el hombre domine sus instintos para equilibrar su capacidad genética con su

capacidad técnica, este modelo aboga por el establecimiento de políticas nacionales de control de la natalidad.

Durante los años setenta, la modelística añade a la relación demografía - desarrollo otras variables, como los recursos y la contaminación. Este sería el modelo de los límites del crecimiento, según el cual el crecimiento de la población acelera el agotamiento de los recursos y aumenta el deterioro ambiental. La naturaleza lleva camino de agotarse como fuente de recursos, siempre en el marco de los actuales parámetros técnicos. De no cambiar estos últimos, las materias primas y la energía básica de que se sirve la actual sociedad, tienen unos límites temporales, a un plazo más o menos largo según diversas perspectivas o escenarios, pero que son inexorables.

Como consecuencia de nuestras actividades, se añade el proceso de contaminación, y se acumulan en el ambiente elementos no reciclados por la técnica, los cuales producen efectos tóxicos sobre el hombre y los restantes seres vivos, comprometiendo, de cara al futuro, la habitabilidad del planeta.

Tanto el sistema político como buena parte del sistema científico han ignorado, rechazado o descartado otros modelos donde no se toma como negativa a la variable población y, en cambio, se

atribuye a la dinámica técnica y social la capacidad de solucionar los problemas. Así, por ejemplo, está el modelo que considera el crecimiento de la población como un estímulo para el crecimiento económico o el cambio social. O el que relaciona positivamente el tamaño espacial de un país y su volumen de población con el desarrollo económico. O bien aquel otro que no admite que la riqueza sea un factor limitado, sino que, porque es producida por personas, puede ser potenciada por el trabajo humano, por las innovaciones tecnológicas y por la imaginación creativa. C. Clark, J. Matras, J.L. Simon o A. Kahn, son los autores más representativos de estos planteamientos.

Por último, el mundo de la sociedad de consumo y los efectos ocasionados en él por la revolución hedonista y contraceptiva, explica la formulación del modelo de la involución demográfica, según el cual los países ricos no aseguran el reemplazo de las generaciones, y por tanto, la de cerebros, y aunque cambien en la década de los noventa la conducta de la mujer blanca en cuanto a la maternidad, el bache a mediano o largo plazo es inevitable, debido a la sobrecarga de las generaciones productivas adultas cercadas por las personas de la tercera edad y por las exiguas generaciones infantiles. Con esta perspectiva, la situación

occidental en los inicios del siglo XXI será la de un yermo en la producción y en la transmisión de cultura.

Además, para que exista una modelo poblacional, debe considerarse las siguientes tasas que dan lugar a su crecimiento:

- **Tasa de fecundidad**

Esta tasa representa cuan fecundada es una población, y no debe confundirse con la tasa de natalidad. La tasa de fecundidad está dada por el número de nacimientos (generalmente anual) por 1000 mujeres en edad de procrear, esta edad generalmente es desde los 15 hasta los 44 años.

- **Tasa de natalidad**

Esta tasa representa la cantidad de nacimientos de una población, y es la cifra bruta de nacimientos por 1000 de la población total por cada año.

- **Tasa de mortalidad**

Esta tasa es representa cuantos fallecimientos ocurren en una población, es decir que es el número de muertes por 1000 personas, generalmente anual.

- **Tasa neta de reproducción**

Promedio de hijas nacidas por mujer si ésta pasa a lo largo de su vida, desde su nacimiento, ajustándose a las tasas de fecundidad y mortalidad por edad de un año dado. Debe considerarse que algunas mujeres pueden morir antes de que terminen sus años de procreación.

3.2.2.1. Crecimiento demográfico

El crecimiento demográfico es el aumento, en un período específico, que generalmente se toma como de un año, del número de personas que viven en un país o una región.

La tasa de crecimiento demográfico depende, además de los movimientos migratorios, de la tasa de natalidad y de la tasa de mortalidad. La tasa de natalidad depende a su vez de la tasa de fecundidad. La tasa de fecundidad está influida por muchos factores pero el principal es el nivel cultural de la sociedad y especialmente de las mujeres: a mayor cultura, menor número de hijos se tiene. La tasa de mortalidad depende del grado de desarrollo económico y sanitario.

Es tal la importancia de la natalidad y la mortalidad en el crecimiento demográfico, que se puede tomar la tasa de

crecimiento demográfico como la diferencia entre las tasas de natalidad y de mortalidad.

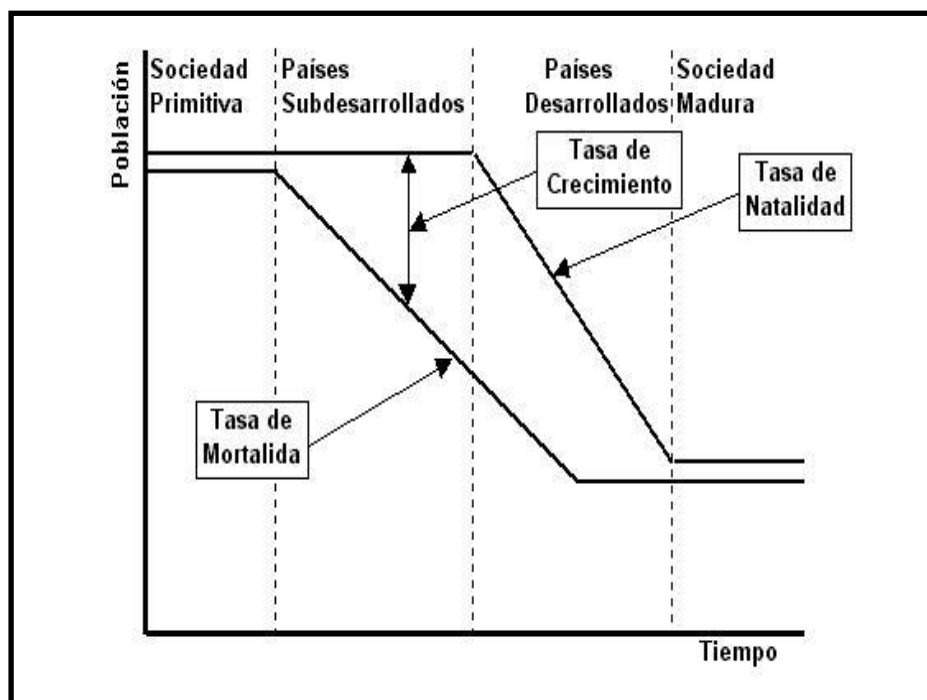


Gráfico de las tasas de natalidad, mortalidad y crecimiento

Figura 3.1

En las sociedades primitivas las tasas de fecundidad y de mortalidad son muy altas pero la diferencia entre ellas es muy estrecha por lo que la población se mantiene estable a largo plazo. Esta fue la situación de todo el mundo hasta hace trescientos años, pero aún siguen existiendo grupos marginales de población con estos parámetros demográficos. En los países subdesarrollados la tasa de mortalidad desciende mucho más rápidamente que las tasas de natalidad y de fecundidad ya que los

avances en la medicina occidental se extienden y se aplican con facilidad mientras que los cambios culturales requieren más tiempo. Como consecuencia de esa disparidad la tasa de crecimiento de la población aumenta de forma explosiva. En los países desarrollados, los cambios culturales y las avanzadas técnicas de control familiar permiten el descenso de la tasa de fecundidad y de crecimiento. Finalmente, las sociedades maduras se caracterizan por tasas demográficas muy bajas y población estable. Esta es la situación actual en los países más desarrollados.

Pero ese esquema no se repite exactamente igual en todos los países. La velocidad de los cambios difiere de un país a otro provocando así grandes disparidades con importantes repercusiones en la distribución de las rentas. En los países europeos, los avances en la medicina se han estado introduciendo paulatinamente durante los últimos doscientos cincuenta años. Los cambios culturales y en la mentalidad evolucionaron de forma paralela, permitiendo un descenso acompasado de la tasa de natalidad. En consecuencia, aunque la tasa de crecimiento de la población ha sido alta en Europa durante mucho tiempo, nunca ha alcanzado las características explosivas típicas de los actuales países subdesarrollados.

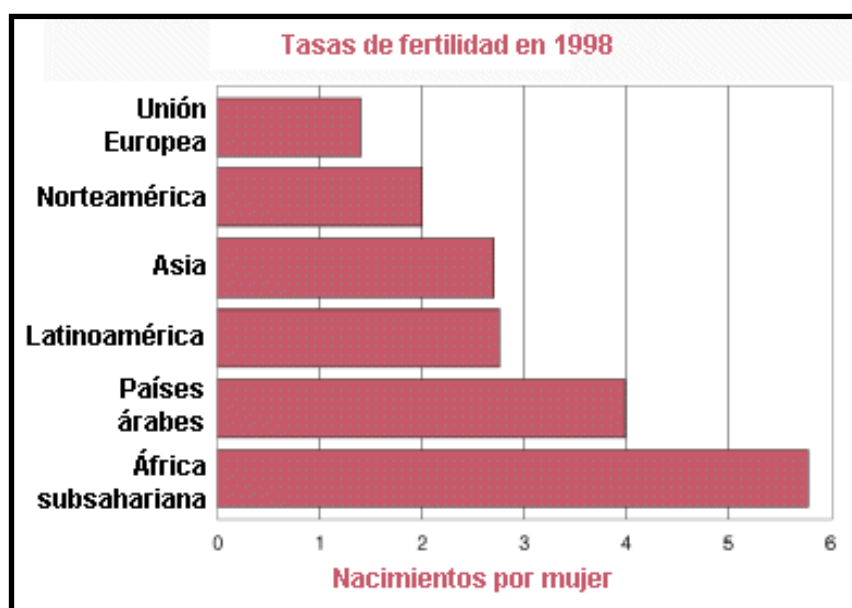


Gráfico de las tasas de fertilidad de 1998

Figura 3.2

En los países avanzados se ha alcanzado el estado de madurez, con la población estabilizada. En algunos países subdesarrollados no existe ningún control de la natalidad por razones políticas, ideológicas o culturales por lo que la población sigue creciendo de forma explosiva, doblándose en menos de veinte años; en algunos países árabes incluso se está fomentando el crecimiento de la población. Sin embargo, en la mayoría de los países subdesarrollados, las campañas a favor del control de la natalidad están consiguiendo reducir las tasas de fecundidad; a pesar de ello, como las generaciones jóvenes que alcanzan la edad fértil son mucho más numerosas que las que les precedieron, las tasas

de crecimiento de la población continuarán muy altas durante algunos decenios más.

3.2.3. Superpoblación y recursos

En este sentido otro neomaltusiano famoso, Paul Erlich, asegura que el problema de la superpoblación no se relaciona con la cantidad o densidad de habitantes, sino con su nivel de consumo. Siguiendo esta línea de pensamiento, sostiene que será más provechoso para el mundo que la cantidad de hijos por matrimonio se redujera en los Estados Unidos y no en Bangladesh.

El impacto ambiental es directamente proporcional a la población, a su consumo y al daño provocado por la tecnología empleada para satisfacer la demanda, afirma Erlich. En términos globales, las naciones ricas, con un cuarto de la población mundial, consumen las dos terceras partes de la energía.

Algunas posturas optimistas, como las de John Holden, profesor de la Universidad de California, sostienen que la ecuación alimentos y población se podrá resolver con 10.000 millones de habitantes consumiendo en promedio 3 kilovatios por cabeza. Si bien este número permitirá a la mayoría de la población triplicar su consumo, el cuarto restante tendrá que reducirlo a menos de la

mitad. De todos modos, se pregunta Erhlich, habrá que ver si el ambiente soportara una demanda de 30 teravatios (10.000 millones x 3 kilovatios), casi el doble del consumo actual. “La clave para entender la superpoblación, - afirman Paul y Anne Erhlich en *The population explosion*- está en la relación entre el número de habitantes en un área determinada y la capacidad de ese ambiente en sostener la actividad humana”.

Cuando un área está superpoblada, la población se mantiene gracias al rápido deterioro de los recursos no renovables, o mediante la conversión de renovables en no renovables, afirman los autores.

Desde ese punto de vista, el planeta mismo ya está superpoblado. Deforestación, desertificación, contaminación de las aguas subterráneas, agotamiento de los acuíferos, sobreexplotación pesquera, calentamiento global serán sólo algunos de los síntomas más visibles de que nos estamos acercando al techo. “El desarrollo económico sostenible es una ilusión en un mundo de recursos limitados”, sostenía Jacques Costeau, y llamaba a la revisión del concepto “progreso” sobre la base de una concepción de lo finito, más que de lo renovable.

3.2.3.1. El cero es una buena nota

¿Cómo es posible detener el aumento de la población? Lo cierto, lo indiscutido, es que hay que hacerlo. Para estabilizar la población, la tasa de crecimiento debe ser cero. Osea dos hijos por pareja o sólo un poco más. Pero cuando se habla de control de la natalidad es inevitable caer en los derechos del hombre a elegir cuántos hijos tener. Ni más ni menos de los que desea. Esto podrá traducirse en: ni métodos coercitivos, ni falta de información sobre planificación familiar.

De un encuentro organizado por la FAO y la UNFPA sobre producción de alimentos y crecimiento poblacional realizado en Roma en julio del '96, surgieron las tres variables a manejar:

- Reducción del número de nacimientos no deseados, eufemismo utilizado para designar a la planificación familiar.
- Incremento del nivel educativo de la mujer (a mayor educación, menor cantidad de hijos).
- Aumento de la edad al tener el primer hijo (lo mismo propone Malthus en 1798).

El Plan de Acción aprobado en la Conferencia de Población y Desarrollo en El Cairo 1994 pretende estabilizar la población mundial en 9800 millones antes del año 2050. Si bien hay datos

que alientan el optimismo, como que la reducción en la tasa de crecimiento fue mayor en Asia que en Europa entre 1990 y 1995, o que en Europa la tasa de crecimiento está muy cercana a cero, no hay que olvidar que se trata de variables ligadas con lo social, lo cultural y lo político. Analizando el Anuario de las Naciones Unidas, llama la atención que las tasas de dos países como India (918 millones) y Pakistán (126 millones), enemigos de hace largo tiempo y con necesidad de un ejército numeroso, clave, sean del 2,4 por ciento y 3,1 por ciento respectivamente. ¿Pura casualidad? También en algún momento ha salido publicado en los diarios la preocupación del gobierno Israel porque la tasa de natalidad de los palestinos es superior a la de los propios israelíes. O de los norteamericanos que ven avanzar a la minoría hispana a grandes pasos.

Finalmente parece cierto que la población sumará 2000 millones de nuevos habitantes para el año 2020; que con bastante esfuerzo la tecnología pondrá lo que falta para comer y que el deterioro del ambiente seguirá aumentando como hasta ahora. Desde el momento en que esto salga publicado hasta el 2020 quedarán menos de 22 años. En ese lapso la estabilización de la población dependerá de los más flojos en cada tema: la cantidad la regularán los países en desarrollo, donde están las tasas de

crecimiento más altas, y el freno al consumo de energía estar a cargo de los pases industrializados, de los habitantes de un país como EE.UU. que consumen 11 kilovatios cada uno.

3.2.4. Modelos de crecimiento poblacional

Las poblaciones experimentan procesos que hacen variar el número de individuos y que cuando se compensan entre sí el número el tamaño se mantiene constante.

La Mortalidad y la natalidad se pueden expresar en tasas porcentuales y diferenciales. Los procesos que hacen cambiar a las poblaciones son: emigración, inmigración, natalidad y mortalidad.

- **Mortalidad:** Número de individuos muertos.
- **Natalidad:** Número de individuos nacidos.

En un periodo de tiempo, con relación al número de individuos al inicio del periodo de tiempo. La natalidad y la mortalidad se dan en forma simultánea y su diferencia mostrará que la población crezca o disminuya.

Cuando las tasas de natalidad o mortalidad se expresan en tasas diferenciales, en un tiempo infinitamente pequeño, se suelen usar las letras:

b : natalidad

m : mortalidad

$$r = b - m$$

Entonces podemos decir que r es el índice reproductor neto o tasa intrínseca de aumento natural en una población.

Efecto de la natalidad: $dN / dt = bN$

Efecto de la mortalidad: $dN / dt = -mN$

Efecto conjunto: $dN / dt = (b - m)N$

$$dN / dt = rN$$

donde : $r = b + (-m)$

$$r = b - m$$

r = tasa intrínseca de aumento natural

El potencial biótico de una especie se define cuando la natalidad se hace máxima y la mortalidad se hace mínima. Obviamente en este caso la población maximizará su crecimiento.

Integrando: $N_t = N_0 e^{rt}$

N_t = Número de individuos en el tiempo t

N₀ = Número inicial de individuos

e = base de logaritmos neperianos

r = índice intrínseco de crecimiento poblacional o tasa intrínseca de aumento natural.

t = intervalo de tiempo.

Si la natalidad supera siempre a la mortalidad, el valor de r será siempre positivo y la población experimentará un crecimiento, sino el valor de r será negativo y se tendrá un decrecimiento poblacional.

Desde el Laboratorio de Ecología de Poblaciones de la Facultad de Ciencias Exactas de la UBA, David Bilenca, investigador y docente, no cree que sea posible trasladar modelos de crecimiento poblacional a la humanidad. “Básicamente hay dos modelos, uno de crecimiento exponencial, en donde la población se comporta como si los recursos fuesen ilimitados, y otro logístico, donde a través de factores de regulación interna se llega a un equilibrio entre la población y la capacidad de carga o soporte del ambiente”, explica el investigador.

En el modelo exponencial la población consume los recursos mientras se multiplica geométricamente hasta agotarlos, momento en el cual la cantidad de individuos cae abruptamente. En cambio en el modelo logístico aparecen mecanismos de autocontrol que estabilizan la población. En este caso se habla de poblaciones densodependientes.

Ambos modelos tienen en común que convergen en un límite, y ese límite existe por tratarse de recursos finitos. En un caso es el ambiente el que fija el límite. En el otro son los mecanismos de autorregulación.

Sin embargo, Bilenca cree que no es válido extrapolar estos modelos al comportamiento de la población humana. En primer lugar porque no hay suficiente información a lo largo del tiempo como para fijar qué tipo de comportamiento tiene. Y en segundo lugar porque entiende que los factores socioculturales intervienen mucho antes que cualquier ley biológica. “Habrá que preguntarse por qué una mujer muy rara vez supera los 12 o 14 hijos, cuando potencialmente est preparada para tener muchos más”, se pregunta dejando una sensación muy extraña antes de despedirse.

A pesar de ser conocido por sus teorías de la población, Thomas Robert Malthus fue sobre todo un pensador interesado por cómo la felicidad (al menos en su sentido material), como un bien escaso, podía ser repartida entre los hombres.

Por primera vez alguien se atreva a decir que tener cada vez más trabajadores, soldados y, por qué no, desocupados, no significaba necesariamente una ventaja para las naciones, sino más bien un

límite a las posibilidades de mejorar la situación de los que ya existían.

Según Malthus, para lograr el bienestar es necesario encontrar “el equilibrio entre la capacidad productiva y el deseo de consumir”. El mundo parece una torta que crece a medida que el cocinero aprende nuevas recetas, pero que depende de la cantidad de comensales para cortar las porciones. Lo mejor será no repartir demasiadas invitaciones, sobre todo cuando la torta ya no puede crecer más.

Aportó como un fundamento teórico para quienes combatan cualquier intento de aliviar el infortunio de los más pobres. Malthus, muerto el 23 de diciembre de 1834, es hoy declarado un enemigo del gasto social.

3.2.4.1. Modelo de crecimiento exponencial:

$$\frac{dN}{dt} = rN$$

Postula que en un ambiente donde no haya factores que limiten, entonces la población crecerá exponencialmente. Por cada unidad de tiempo que pase la población se multiplicara por una

cantidad constante, mientras más grande sea el valor de N , entonces mayor será el crecimiento.

Para este modelo haremos los siguientes supuestos:

- El crecimiento no tiene límites.
- La tasa de crecimiento poblacional (r) es constante.
- Todos los individuos de la población son idénticos, no se toman en cuenta las edades o sexo. Es evidente que ninguna población puede crecer en forma exponencial indefinidamente ya que en algún momento los recursos se agotan.

Las poblaciones r estrategias (pioneros u oportunistas) que están seleccionadas por su rápido crecimiento (r seleccionadas) se ajustan a un modelo de crecimiento.

Es fácil expresar el tamaño de la población como una ecuación de la forma:

$$P_{(t)} = P_0 e^{ax}$$

Donde a es un parámetro propio del modelo, t es el t -ésimo año, P_0 es el tamaño de la población en el año cero, y $P_{(t)}$ es la función evaluada en el año t -ésimo. Entonces escribiré la ecuación anterior de la siguiente forma:

$$Y = \beta_0 e^{\beta_1 x}$$

Donde β_0 y β_1 son los parámetros del modelo, t es el t -ésimo año y Y es la función evaluada en el año t -ésimo. Donde hallaré los valores para β_0 y β_1 ; pero para realizar la regresión con éste modelo, debe hacerse una conversión mediante el uso de logaritmos naturales en ambos lados de la ecuación, obteniéndose el siguiente modelo:

$$\ln(Y) = \ln(\beta_0) + \beta_1 x$$

Además, para realizar la regresión en este modelo, puede hacérselo de la misma forma que en la regresión lineal, donde se estiman los β_i de la siguiente forma:

$$\hat{\beta} = (X^t X)^{-1} X^t Y$$

Donde la matriz Y será de la siguiente forma:

$$Y = \begin{bmatrix} \ln(Y_1) \\ \ln(Y_2) \\ \dots \\ \ln(Y_n) \end{bmatrix}$$

Donde Y_1 es el primer valor observado de $Y=P_{(t)}$, equivalente a P_0 , Y_2 es el segundo valor observado de $Y=P_{(t)}$, y Y_n es el n-ésimo valor observado de $Y=P_{(t)}$.

La matriz X será:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ \cdot & \cdot \\ 1 & n \end{bmatrix}$$

Donde n es el número de datos u observaciones, que en este modelo representa el número de años que se desea hacer el seguimiento al modelo.

Entonces evidentemente la matriz X^tY será de la siguiente forma:

$$X^tY = \begin{bmatrix} \ln(Y_1) + \ln(Y_2) + \dots + \ln(Y_n) \\ \ln(Y_1) + 2\ln(Y_2) + \dots + n\ln(Y_n) \end{bmatrix}$$

La cual puede escribirse como:

$$X^tY = \begin{bmatrix} \sum_{i=1}^n \ln(Y_i) \\ \sum_{i=1}^n i \ln(Y_i) \end{bmatrix}$$

Por otra parte, al realizar la multiplicación de las matrices $X^t X$ se obtiene la siguiente matriz:

$$X^t X = \begin{bmatrix} n & \frac{n(n+1)}{2} \\ \frac{n(n+1)}{2} & \frac{n(n+1)(2n+1)}{6} \end{bmatrix}$$

Entonces, haciendo uso de la inversa para matrices de dimensión 2x2, se obtiene la inversa de la matriz $X^t X$ de la siguiente forma:

$$\left(X^t X\right)^{-1} = \frac{1}{\det\left(X^t X\right)} \begin{bmatrix} \frac{n(n+1)(2n+1)}{6} & -\frac{n(n+1)}{2} \\ -\frac{n(n+1)}{2} & n \end{bmatrix}$$

La cual puede escribirse de la siguiente manera:

$$\left(X^t X\right)^{-1} = \left(\frac{(n^2)(n+1)(2n+1)}{6} - \frac{(n^2)(n+1)^2}{2}\right)^{-1} \begin{bmatrix} \frac{n(n+1)(2n+1)}{6} & -\frac{n(n+1)}{2} \\ -\frac{n(n+1)}{2} & n \end{bmatrix}$$

El vector de los β tendrá la forma siguiente:

$$\hat{\beta} = \begin{bmatrix} \ln(\beta_0) \\ \beta_1 \end{bmatrix}$$

Entonces la regresión queda de la forma:

$$\begin{bmatrix} \ln(\beta_0) \\ \beta_1 \end{bmatrix} = \left(\frac{(n^2)(n+1)(2n+1)}{6} - \frac{(n^2)(n+1)^2}{2} \right)^{-1} \begin{bmatrix} \frac{n(n+1)(2n+1)}{6} & -\frac{n(n+1)}{2} \\ -\frac{n(n+1)}{2} & n \end{bmatrix} \begin{bmatrix} \sum_{i=1}^n Y_i \\ \sum_{i=1}^n iY_i \end{bmatrix}$$

3.2.4.2. Modelo logístico de crecimiento

Supone que la población no crece indefinidamente y mientras mayor sea su densidad más lento será el crecimiento, y se detendrá cuando la población alcance un límite denominado capacidad de carga.

Experimenta retroalimentación negativa, la población crece solo hasta un límite llamado capacidad de carga, y cuando se supera este límite, se disminuye su tamaño.

Una población esta influenciada en su mayor o menor grado por el medio ambiente, esto manifiesta en el tamaño y el crecimiento de la población.

El punto máximo de población es el punto K, es decir el número de población máxima que alcanza la por la limitación de recursos que generalmente es el alimento.

$$\frac{dN}{dt} = rN \left[\frac{k-N}{k} \right]$$

1. El crecimiento tiene límites.
2. El efecto de sumar o de restar un individuo a la población es registrado de forma inmediata modificando por lo tanto, la tasa de crecimiento en forma instantánea. No se complementa tiempos de reparto.
3. Todos los individuos de la población son idénticos. Esto significa que no se toman en cuenta las edades o sexo.
4. La tasa de crecimiento poblacional (r) no es constante es una función lineal de la densidad poblacional.

Cuando un número de individuos es máximo, r es mínimo, y cuando el número de individuos es mínimo r es máxima.

En realidad este modelo es una generalización del modelo exponencial, en el cual se considera un límite de crecimiento, el tamaño de la población puede escribirse como:

$$P_{(t)} = \frac{aP_0}{bP_0 + (a - bP_0)e^{-a(t-t_0)}}$$

Donde a y b son constantes propias del modelo, t es el tiempo, t_0 es el tiempo inicial de la población, P_0 es el tamaño inicial de la población, $P_{(t)}$ es el tamaño de la población en el instante t . Aunque también puede escribirse como:

$$P_{(t)} = \frac{1}{\frac{1}{U} + (ab^t)}$$

Donde U es el punto máximo al que puede llegar a crecer la población, y debe ser mayor a todos los datos observados, a y b son los parámetros propios del modelo.

Para tener una mejor apreciación del modelo, puede escribirse de la forma:

$$Y = \frac{1}{\frac{1}{U} + (\beta_0 \beta_1^x)}$$

Entonces, hallando la inversa en ambos lados de la ecuación y luego tomando los logaritmos naturales, puede escribirse el modelo de la siguiente manera:

$$\ln\left(\frac{1}{Y} - \frac{1}{U}\right) = \ln(\beta_0) + \ln(\beta_1)x$$

Al tener el modelo de esta forma, puede plantearse de igual manera que en el caso anterior para poder hacer una regresión mediante el método de los mínimos cuadrados:

$$\hat{\beta} = (X^t X)^{-1} X^t Y$$

Donde la matriz Y será de la siguiente forma:

$$Y = \begin{bmatrix} \ln\left(\frac{1}{Y_1} - \frac{1}{U}\right) \\ \ln\left(\frac{1}{Y_2} - \frac{1}{U}\right) \\ \vdots \\ \ln\left(\frac{1}{Y_n} - \frac{1}{U}\right) \end{bmatrix}$$

Donde Y_1 es el primer valor observado de $Y=P_{(t)}$, equivalente a P_0 , Y_2 es el segundo valor observado de $Y=P_{(t)}$, y Y_n es el n-ésimo valor observado de $Y=P_{(t)}$.

La matriz X será:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ \cdot & \cdot \\ 1 & n \end{bmatrix}$$

Donde n es el número de datos u observaciones, que en este modelo representa el número de años que se desea hacer el seguimiento al modelo.

Entonces evidentemente la matriz X^tY será de la siguiente forma:

$$X^tY = \begin{bmatrix} \sum_{i=1}^n \left(\frac{1}{Y_i} - \frac{1}{U} \right) \\ \sum_{i=1}^n i \left(\frac{1}{Y_i} - \frac{1}{U} \right) \end{bmatrix}$$

Por otra parte, al realizar la multiplicación de las matrices X^tX se obtiene la siguiente matriz:

$$X^tX = \begin{bmatrix} n & \frac{n(n+1)}{2} \\ \frac{n(n+1)}{2} & \frac{n(n+1)(2n+1)}{6} \end{bmatrix}$$

Entonces, haciendo uso de la inversa para matrices de dimensión 2×2 , se obtiene la inversa de la matriz X^tX de la siguiente forma:

$$\left(X^tX \right)^{-1} = \frac{1}{\det(X^tX)} \begin{bmatrix} \frac{n(n+1)(2n+1)}{6} & -\frac{n(n+1)}{2} \\ -\frac{n(n+1)}{2} & n \end{bmatrix}$$

La cual puede escribirse de la siguiente manera:

$$(X^t X)^{-1} = \left(\frac{(n^2)(n+1)(2n+1)}{6} - \frac{(n^2)(n+1)^2}{2} \right)^{-1} \begin{bmatrix} \frac{n(n+1)(2n+1)}{6} & -\frac{n(n+1)}{2} \\ -\frac{n(n+1)}{2} & n \end{bmatrix}$$

El vector de los β tendrá la forma siguiente:

$$\hat{\beta} = \begin{bmatrix} \ln(\beta_0) \\ \ln(\beta)_1 \end{bmatrix}$$

Entonces la regresión queda de la forma:

$$\begin{bmatrix} \ln(\beta_0) \\ \ln(\beta_1) \end{bmatrix} = \left(\frac{(n^2)(n+1)(2n+1)}{6} - \frac{(n^2)(n+1)^2}{2} \right)^{-1} \begin{bmatrix} \frac{n(n+1)(2n+1)}{6} & -\frac{n(n+1)}{2} \\ -\frac{n(n+1)}{2} & n \end{bmatrix} \begin{bmatrix} \sum_{i=1}^n \left(\frac{1}{Y_i} - \frac{1}{U} \right) \\ \sum_{i=1}^n i \left(\frac{1}{Y_i} - \frac{1}{U} \right) \end{bmatrix}$$

3.2.4.3. Población en crecimiento y población en retracción.

Llega un momento en que las poblaciones son plurianuales, que a la población se le terminan los recursos que están en ambientes estables para su crecimiento y se mantiene. Esta situación hace que en la población tenga que mantenerse o tenga que decrecer. Son autorregulables y se tornan en un modelo de tipo logístico.

3.2.4.4. Características Comparativas de los modelos exponencial y logístico

	r estrategias	k estrategias.
Ambiente:	Variable y/o impredecible.	Constante y/o predecible.
Mortalidad:	Densodependientes.	Densoindependiente.
Tamaño poblacional:	Variable sin llegar al k.	En equilibrio o tendiendo a k.
Competencia:	Variable en general laxa.	Fuerte.
Rasgos:	Positivos.	Positivos.
Crecimiento:	Rápido (r alta).	Lento.
Competitividad:	Poco competitivo.	Muy competitivo.
Reproducción:	Temprana.	Tardía (retrasada).
Nº de reproducción:	Unico.	Reiterado.
Tamaño corporal:	Pequeño.	Grande.
Descendencia:	Muy Pequeña.	Pocos grandes
Ciclo de vida:	Corto.	Largo.
Etapas serial	Temprana	Tardío.

Tabla de características comparativas entre los modelos exponencial y logístico

Tabla I

En la realidad en pocas ocasiones el crecimiento poblacional se ajusta a alguno de los modelos. Estos modelos son determinísticos y suponen que no hay gran variación entre los individuos de una población. En realidad el crecimiento poblacional puede ser mejor explicado por modelos estocásticos. Los modelos estocásticos están basados en la probabilidad de variaciones dentro de una población. Son mucho más complejos matemáticamente porque incorporan otros factores, tales como los procesos de Markov, cadenas de Markov, procesos de Poisson, simulación bajo situación de incertidumbre, entre otros.

A pesar de la dificultad para predecir de manera precisa crecimiento poblacional, podemos hacer algunas generalizaciones:

1. Existe una correlación entre tamaño de la población y crecimiento poblacional, al menos que se apliquen criterios para evitar el crecimiento de la población.
2. Los organismos que tardan más en llegar a la edad de reproducirse tienen tasas menores de crecimiento poblacional.

3.2.4.5. Modelo poblacional utilizando los algoritmos genéticos

Antes de hablar de un modelo con algoritmos genéticos, debe recordarse que la ventaja de los algoritmos genéticos como herramienta para cualquier aplicación es el hecho de poder trabajar con la misma función propuesta, y no con una derivada o cualquier otro conocimiento matemático.

Para modelar el crecimiento de una población utilizando los algoritmos genéticos como la herramienta del modelo, debe estimarse correctamente las variables que se mencionaron anteriormente, que son los índices de natalidad, mortalidad, y las migraciones, además debe considerarse el número de hijos que

una persona tendrá durante su vida y el intervalo de edades en que una persona está apta para reproducirse.

Mi idea no es desarrollar un modelo matemático, ni un modelo completamente sencillo, ni homogéneo, sino que deseo utilizar la simulación para determinar el crecimiento poblacional, pero esta simulación está basada en el paradigma de los algoritmos genéticos.

La aplicación es desarrollada completamente en Visual C++ 6.0, en el modo MFC bajo Windows en 32 bits. Para facilitar el trabajo, la aplicación no utiliza ninguna base de datos, sino que trabaja utilizando archivos directos, los cuales reciben las siguientes extensiones:

.Agc Extensión de los archivos de ingreso

.Agr Extensión de los archivos de resultados

.Agd Extensión de los archivos temporales de datos

.Agm Extensión de los archivos de datos del Optimizador de los valores de los Betas.

.Agp Extensión de los archivos de datos del Seleccionador de modelos.

.Dat Extensión de los archivos de datos privados de la aplicación.

Para el diseño del modelo, he utilizado una clase llamada AGpob, que contiene las estructuras necesarias para cumplir con el objetivo de modelar el crecimiento poblacional mediante los algoritmos genéticos. Esta clase cuenta esencialmente con dos listas enlazadas que representan la población discriminada en dos grupos que son hombres y mujeres.

Cada hombre o mujer de la población son representados mediante una estructura de tipo persona, la cual contiene a la vez dos estructuras, la de tipo genético y la de tipo datos de la persona. La estructura de tipo genético contiene todas las partes necesarias del algoritmo genético que son:

- ⌘ Los genes o bits del cromosoma, que en este caso fueron utilizados de tipo caracter, aunque podrían ser enteros.
- ⌘ Los cromosomas o individuos de la población, que son cadenas de caracteres binarios de longitud determinada.
- ⌘ El genotipo, que es la representación decimal del cromosoma o individuo dentro de un intervalo que es el dominio de la función de aptitud.
- ⌘ La aptitud, que es el genotipo evaluado en la función de aptitud y que sirve para determinar cuando un individuo está más apto que otro para reproducirse.

§ La disposición o aptitud relativa, que es la frecuencia relativa de la función de aptitud.

Así mismo, la estructura de datos de la persona contiene datos esenciales para un modelo poblacional, que son:

- ★ La edad actual del individuo, que al nacer es 0, y luego se incrementará.
- ★ La edad límite del individuo, que es un número aleatorio de la función propuesta para la mortalidad.
- ★ La edad inicial y final de reproducción que son valores aleatorios de la función escogida para la reproducción.
- ★ El número total de hijos que tendrá en toda su vida, que es un valor aleatorio.
- ★ El número actual de hijos, que es un contador.

Por otra parte, debe considerarse una correcta función de selección y de adaptación o aptitud de los algoritmos genéticos, ya que Darwin introdujo la adaptación en la teoría evolutiva mediante un cuarto principio: “Las variaciones que favorecen la supervivencia de un individuo en competencia con otros organismos, y a pesar de la tensión ambiental, tienden a aumentar el éxito reproductivo, y por tanto, tienden a conservarse (principio de la lucha por la existencia)”.

Al considerar que la adaptación resulta de la selección natural bajo la presión de la lucha por la existencia, se la sobreentiende como una condición relativa, no como una condición absoluta. Aun cuando una especie pueda sobrevivir en gran número, y por consiguiente pueda estar adaptada en un sentido absoluto, cabe la posibilidad de que surja una nueva forma que tenga un ritmo reproductivo mayor con los mismos recursos, y ocasione la extinción de la forma antigua.

Es por eso que la función de selección la escogí basándome en las edades en las que un individuo puede reproducirse, y en la aptitud que éste puede tener para reproducirse. Los métodos de selección que utilicé fueron el método del torneo y el método de la rueda de la ruleta, dándose además la posibilidad de mantener el estado y de utilizar elitismo para el crossover.

La función de aptitud que utilicé está basada en la diversificación de recursos de la población, es por eso que la escogí como una función que tenga un crecimiento muy lento, para dar oportunidad a todos los individuos a desarrollarse. He considerado la raíz cuadrada como una función bastante buena en este sentido, esta función de aptitud es equivalente a los recursos con los que cuenta cada individuo para sobrevivir en la población.

De igual manera, es importante entender lo que representan el crossover y la mutación en un modelo poblacional elaborado mediante algoritmos genéticos, por lo que decidí que la reproducción de la población se realizaría en el momento del crossover, sin reemplazar la población anterior, sino que el nuevo individuo se agrega a la población. Para el crossover utilicé los métodos de crossover en un punto y crossover uniforme, generando inicialmente catorce posibles cromosomas hijos mediante el crossover por lógica matemática, y luego escogiendo dos de estos aleatoriamente, para realizar el crossover propuesto y generar el nuevo individuo.

Además se da la posibilidad de realizar el crossover solamente en los cromosomas de mayor aptitud, o de realizarlo solamente en los extremos del cromosoma.

En el momento que el individuo es agregado a la población, ya tiene parámetros propios, tales como su edad límite, número de hijos que tendrá y el intervalo de edades en las que se reproducirá.

Ya que el número promedio de hijos es ingresado en el algoritmo, decidí número de hijos que un individuo puede tener, estará dado

como el entero mayor de un valor aleatorio de una población normal con media n y con varianza $n^2/9$, para cubrir casi todo el rango de generación de números aleatorios normales.

Para el intervalo de edades de reproducción, debe realizarse el ingreso del extremo a y del extremo b ; pero en este caso se utilizó valores aleatorios uniformes discretos para cada uno de los extremos, validando que el valor de a sea menor que el de b al momento del ingreso y de la generación de la variable aleatoria.

En cuanto a la función de mortalidad que se utilizará en el proyecto, debe realizarse el ingreso del punto límite de vida (w), y luego se utilizará una función de mortalidad donde la probabilidad de fallecer es directamente proporcional a la edad. Es decir que la persona tiene mayor probabilidad de vivir w años que vivir 0 años, es por eso que la escogí como un valor aleatorio de una función discreta lineal, que es la suma de los primeros n edades de la siguiente manera:

El número aleatorio tiempo de vida está dado por la siguiente función que depende del límite de vida:

$$X = \frac{\sqrt{1 + 4 \cdot \text{Aleatorio}\left(\frac{w \cdot (w + 1)}{2}\right)} - 1}{2}$$

Además esta función da mayor probabilidad a los números mayores.

La mutación representa las posibles enfermedades o curas de las mismas que se presentan en cualquier momento de la vida, y pueden alterar la aptitud de un individuo para reproducirse. La mutación se puede realiza en todos los individuos de la población, sin importar su edad.

Para este proyecto utilicé la mutación en un punto y la mutación uniforme, dando la opción de utilizar la mutación solamente los individuos de menor aptitud y la posibilidad de utilizar la mutación obligada o exigida en cada iteración.

Me encontré en la necesidad de agregar dos funciones al algoritmo genético, que son la eliminación de los ancianos que llegan a la edad límite, para ocasionando la mortalidad, y otra que realice el transcurso del tiempo en cada individuo de la población, para que se incremente su edad actual.

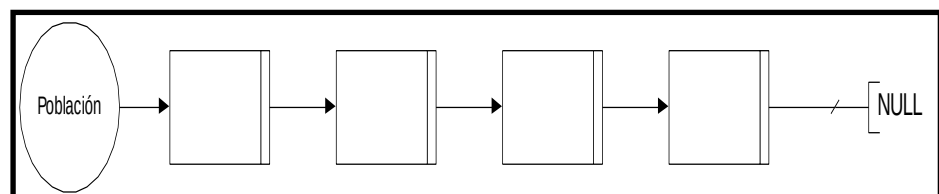
Entonces, podría describirse el modelo de la siguiente manera:

	Estrategias
Ambiente:	Aleatorio.
Mortalidad:	Densodependiente.
Competencia:	Aptitud.
Crecimiento:	Medio, estocástico.
Competitividad:	Modelo de selección.
Reproducción:	Intervalo de fecundidad
Nº de reproducción:	Variable Normal.
Ciclo de vida:	Variable de Moivre

Características del modelo poblacional con algoritmos genéticos

Tabla II

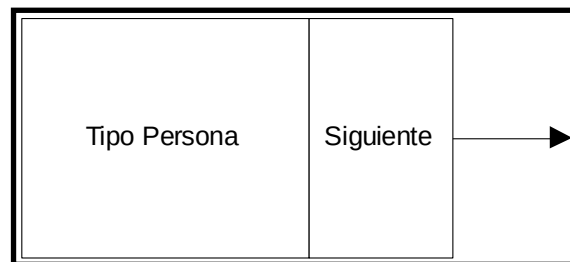
Gráficamente podemos representar el modelo, tanto la estructura como el algoritmo genético aplicado al modelo poblacional como el de la siguiente manera:



Población Separada en hombres y mujeres con n personas.

Figura 3.3

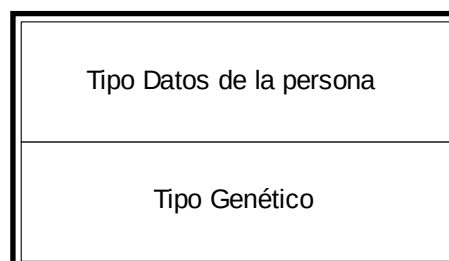
He creado dos listas simplemente enlazadas para representar la población discriminada y separada entre hombres y mujeres



Estructura de tipo población

Figura 3.4

Cada individuo de la población pertenece a la misma, por lo cual, he utilizado una estructura de tipo población, que contiene una estructura de tipo persona y un puntero al siguiente elemento de la lista, que también es de tipo población.



Estructura de tipo persona

Figura 3.5

Cada persona que pertenece a la población se ha trabajado en dos aspectos diferentes que son la parte del algoritmo genético y la parte de los parámetros del modelo poblacional, por lo cual

contiene una estructura de tipo datos de la persona y otra de tipo genético.

Edad Actual	Edad Límite
Edad inicial de Reproducción	Número de hijos
Edad final de Reproducción	Número total de hijos

Estructura de tipo datos de persona

Figura 3.6

Los datos de la persona son los que hacen que el individuo se compare con los parámetros poblacionales, entre ellos se encuentran la edad actual y la edad límite, así como el intervalo de edades en que un individuo es fecundo, y el número de hijos actual, y el máximo número de hijos que tendrá.

Cromosoma	Cromosoma 2
Adaptación	
Genotipo	Disposición

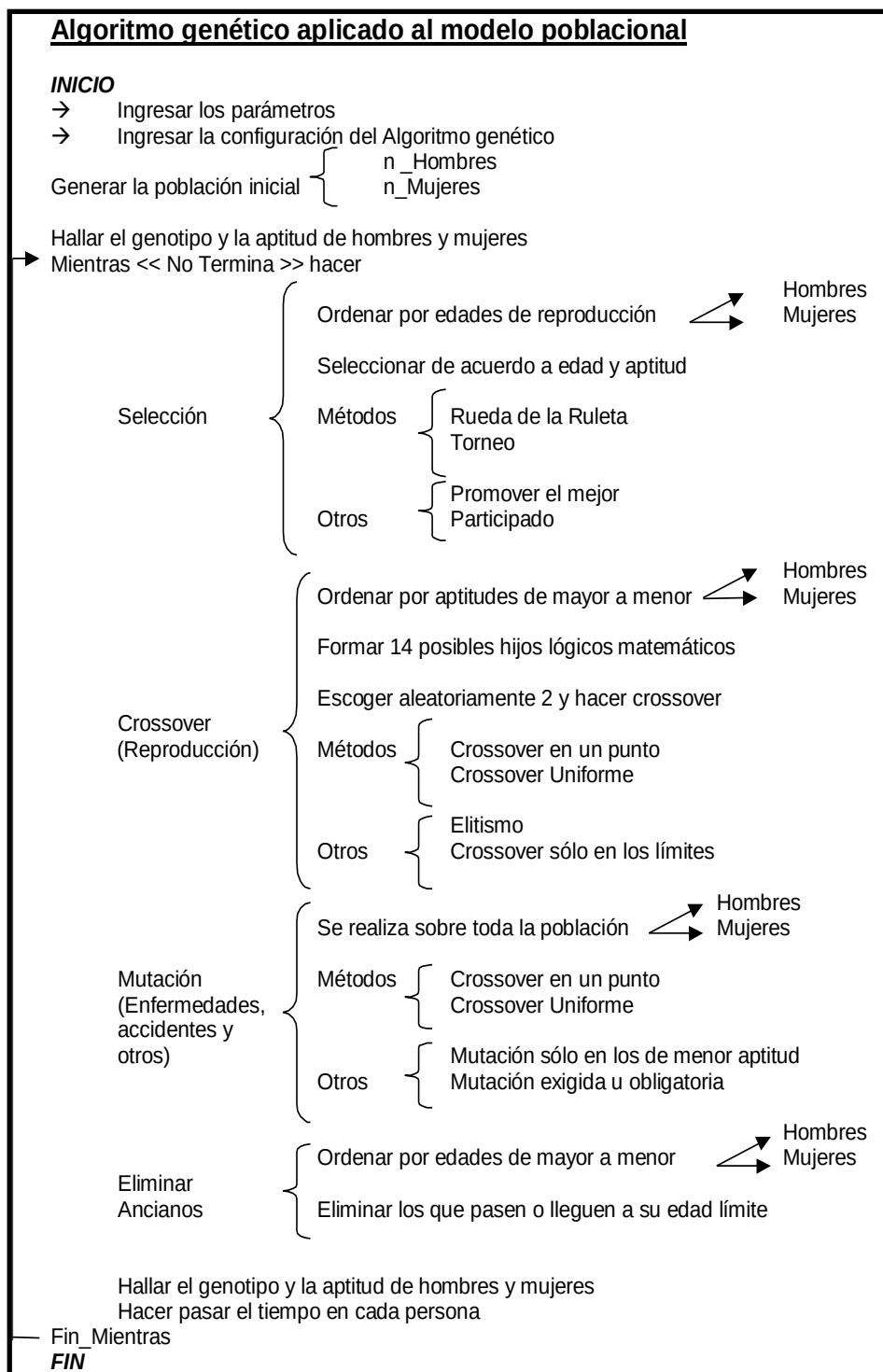
Estructura de Tipo Genético

Figura 3.7

Los componentes o campos de la estructura de tipo genético son el cromosoma principal, el cromosoma 2 que lo utilizo como un auxiliar en la selección y en la reproducción, la adaptación que es el valor que mide los recursos de los individuos, genotipo, que es la base decimal del cromosoma o cadena binaria y la disposición, que es el valor relativo de la aptitud del individuo.

Además, en el siguiente gráfico, muestro un esquema general de los algoritmos genéticos aplicados a un modelo de crecimiento poblacional, en el cual se relacionan cada una de sus partes con las del algoritmo genético básico.

Cabe indicar que es solamente un esquema generalizado, ya que la aplicación presenta muchísimas opciones más, que no han sido utilizadas en el siguiente diagrama de flujo:



Algoritmo genético aplicado a los modelos poblacionales

Cuadro 3.1

CAPITULO IV

4. Simulación poblacional con algoritmos genéticos

4.1. Introducción.

En este capítulo realizo una aplicación robusta de los algoritmos genéticos, utilizándola como la herramienta de simulación de un modelo de crecimiento poblacional, así como dos aplicaciones que determinan cuan bien se ajustan un grupo de datos a un modelo cuyos parámetros son determinados mediante los algoritmos genéticos. Es en este capítulo en el que llevo a la práctica todo lo aprendido durante esta investigación en los capítulos anteriores, implantando una aplicación que sustente mis ideas.

He utilizado el mayor número posible de opciones de los algoritmos genéticos y de los modelos poblacionales, que fueron previamente explicadas en los capítulos 2 y 3.

El modelo poblacional desarrollado con algoritmos genéticos, es un modelo no homogéneo, es decir que existen diferencias entre los individuos de la población, tales como sexo y edad. Estas dos variables ocasionan las variaciones en el modelo, ya que los individuos tienen un límite de vida, y por lo tanto existe una tasa de mortalidad, así como un intervalo de edades en el cual son fecundos, lo que resulta una tasa de fecundidad, y tienen un número de hijos durante toda su vida, lo que da una tasa de natalidad por cada año.

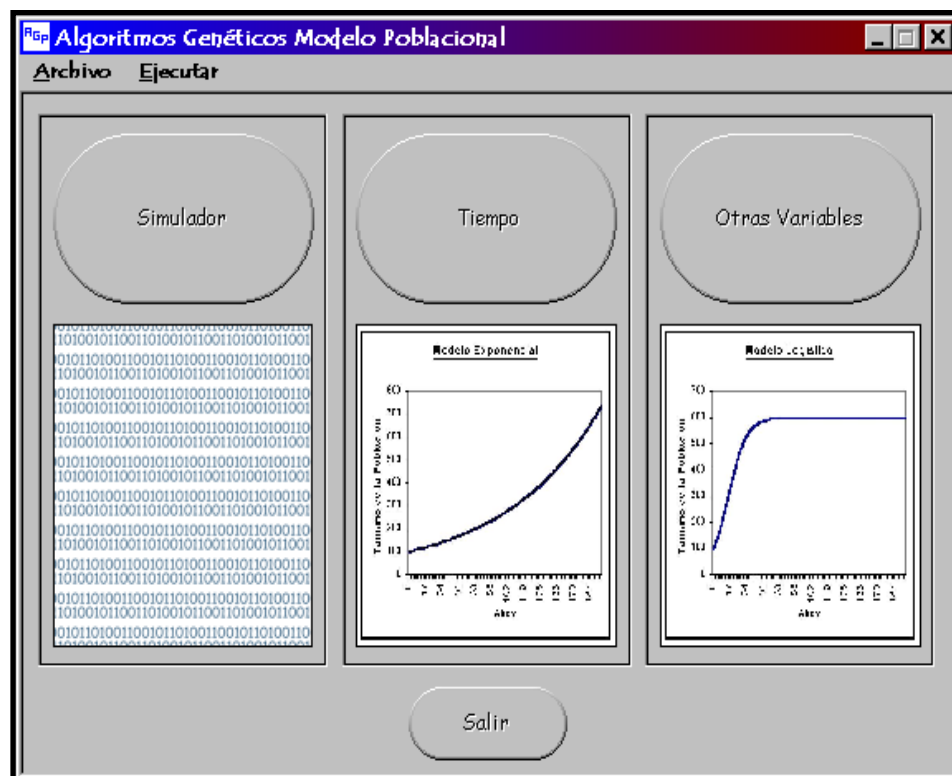
Los algoritmos genéticos se encargan de la parte de selección, reproducción y adaptación de los individuos en el modelo poblacional.

Por otra parte, al hallar una tendencia mediante los algoritmos genéticos para los modelos poblacionales de tipo exponencial y logístico, dejo una idea para una futura investigación acerca de técnicas de regresión mediante el uso de los algoritmos genéticos, aunque también hallo y muestro cual es el mejor modelo del grupo de posibles modelos, utilizando los algoritmos genéticos.

4.2. Manual del usuario.

Para ingresar a la aplicación de Algoritmos Genéticos – Modelo Poblacional, usted deberá ejecutarla con el nombre de AgMp.exe, donde aparecerá una pantalla con tres botones en la parte superior, los mismos que deberá pulsar para ingresar a cualquiera de las tres aplicaciones que he diseñado, y que son el simulador mediante algoritmos genéticos, así como una parte de la aplicación para comparar datos existentes con los modelos exponencial o logístico, ambos diseñados mediante algoritmos genéticos. Además desarrollé una tercera aplicación, una tercera parte que se encarga de seleccionar el mejor modelo de todos los posibles modelos.

Para salir de la aplicación debe pulsar con el ratón el botón Salir, y si desea algún tipo de ayuda, debe elegir el botón ayuda que se encuentra en el mismo cuadro de diálogo.



Pantalla inicial de la aplicación AgMp de Algoritmos Genéticos – Modelo Poblacional

Figura 4.1

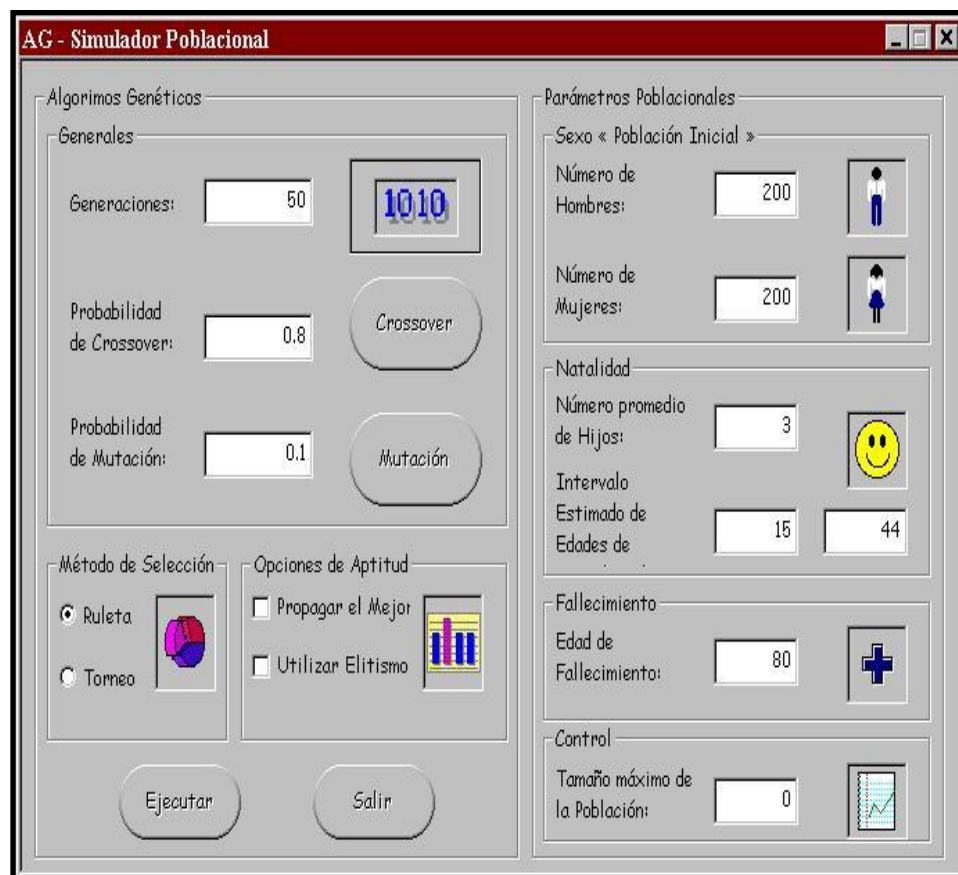
Al ingresar al Simulador del Algoritmos genéticos - modelo poblacional, usted verá una pantalla donde deberá realizar los ingresos, que al lado izquierdo se encuentra la parte de los algoritmos genéticos, y a la derecha la parte de los parámetros del modelo poblacional.

Deberá ingresar el número de generaciones o iteraciones a realizar, la probabilidad de crossover y de mutación, aunque

además puede escoger el tipo junto con otras opciones. Además debe escoger el tipo de selección a utilizar y si desea utilizar elitismo y si desea propagar el elemento con mayor aptitud.

En la parte derecha, debe realizarse el ingreso del número de hombres y el número de mujeres inicial, aunque al iniciar la simulación esos números aparecerán un poco menor.

Debe ingresar el número promedio de hijos y las edades de fecundidad de los individuos, así como la edad límite de vida y el número máximo de individuos que puede haber en la población. En esta última opción el valor de cero significa que no hay límites.



Pantalla Principal del Simulador del AgMp, Algoritmos Genéticos – Modelo Poblacional

Figura 4.2

Usted debe ingresar los datos correctamente y luego presionar el botón ejecutar, con lo que se ejecutará el algoritmo de simulación. Esta parte puede llevarse unos cuantos segundos o minutos, dependiendo de la cantidad de individuos que se tenga en cada iteración.



Línea de progreso de la ejecución del Algoritmos Genéticos – Modelo

Poblacional

Figura 4.3

Al terminar la ejecución, aparecerá una pantalla con los resultados obtenidos en la simulación, los cuales solamente pueden ser observados, y quedarán automáticamente guardados en un archivo de texto llamado Resul1.Agr. En el lado izquierdo usted observará en cada año o iteración de la ejecución, un crecimiento o decrecimiento de hombres y mujeres, así como el total, y el crecimiento o decrecimiento porcentual en cada año.

En la parte derecha de esta pantalla aparecen de igual manera, la mejor aptitud y la aptitud promedio del grupo de individuos. Además se puede ver siete botones en la parte inferior que le permitirán a usted visualizar las gráficas del tamaño poblacional, el crecimiento y decrecimiento porcentual, y de las aptitudes de la población a través del tiempo o guardar o imprimir los datos.

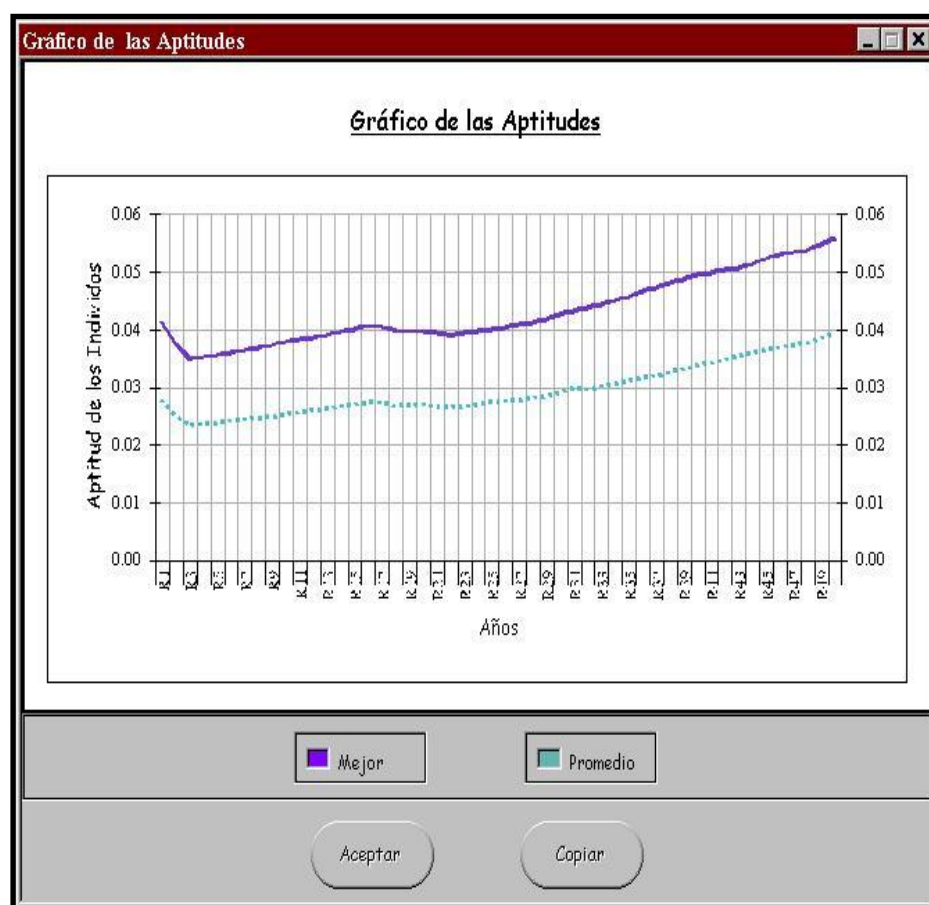
Resultados de la Simulación								
Año	Hombres	Mujeres	Total	Crecimiento	Año	M. Aptitud	Aptitud P.	Edad P.
00	192	195	387	+0.000000	00	0.019687	0.013012	27.49
01	239	239	478	+0.235142	01	0.017715	0.011604	22.74
02	249	254	503	+0.052301	02	0.017269	0.011139	22.25
03	250	256	506	+0.005964	03	0.017218	0.011173	22.88
04	252	256	508	+0.003953	04	0.017184	0.011193	22.74
05	249	254	503	-0.009843	05	0.017269	0.011304	23.37
06	250	253	503	+0.000000	06	0.017269	0.011398	23.91
07	247	253	500	-0.005964	07	0.017321	0.011479	24.27
08	248	251	499	-0.002000	08	0.017338	0.011435	24.35
09	246	249	495	-0.008016	09	0.017408	0.011460	24.66
10	249	246	495	+0.000000	10	0.017408	0.011467	25.06
11	252	242	494	-0.002020	11	0.017425	0.011441	25.32
12	255	246	501	+0.014170	12	0.017303	0.011497	25.36
13	259	247	506	+0.009980	13	0.017218	0.011448	25.59
14	257	247	504	-0.003953	14	0.017252	0.011490	26.10
15	264	247	511	+0.013889	15	0.017133	0.011467	25.66
16	270	257	527	+0.031311	16	0.016871	0.011142	24.82
17	283	267	550	+0.043643	17	0.016514	0.010798	24.18

Aceptar
Tamaño
Crecimiento
Aptitudes
Edades
Guardar
Imprimir

**Pantalla de los resultados del Simulador de los Algoritmos Genéticos –
Modelo Poblacional**

Figura 4.4

En la pantalla del gráfico del tamaño de la población usted puede ver un crecimiento o decrecimiento del tamaño de la población, tanto en hombres como en mujeres, y en su total. En esta pantalla aparecen dos botones, uno de ellos le permitirá copiar la imagen al portapapeles para poder ser tratada posteriormente en otra aplicación, y el otro botón es el que dice aceptar y le permitirá salir de la misma.



Pantalla con el Gráfico del tamaño de la población a través del tiempo

Figura 4.5

En la pantalla del gráfico de las aptitudes, usted puede observar la mejor aptitud del grupo de individuos, así como la aptitud promedio de los individuos. En esta pantalla, usted solamente puede dar un pulso con el ratón en el botón aceptar para poder salir de la misma, o puede hacer un pulso con el ratón en el otro botón para copiar el gráfico al portapapeles para un tratamiento posterior.

Debe recordarse además que la aptitud de los individuos es lo que permite seleccionarlos del grupo dando lugar a que un individuo sea más apto que otro para la reproducción.

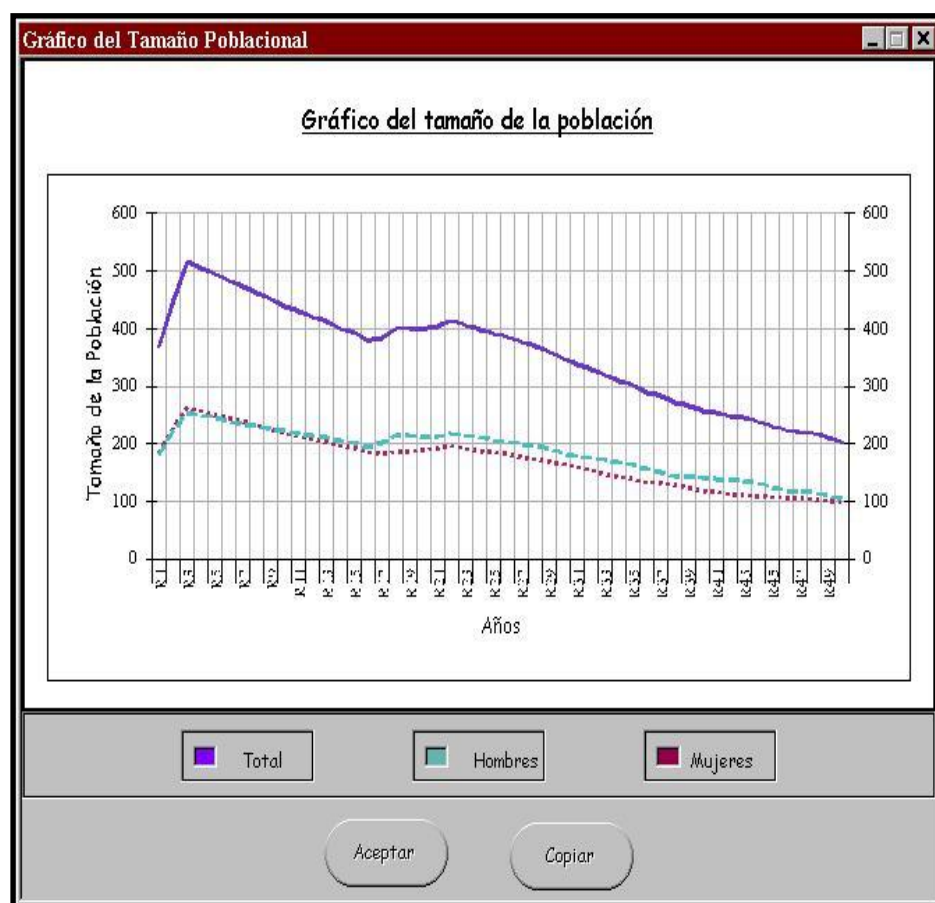
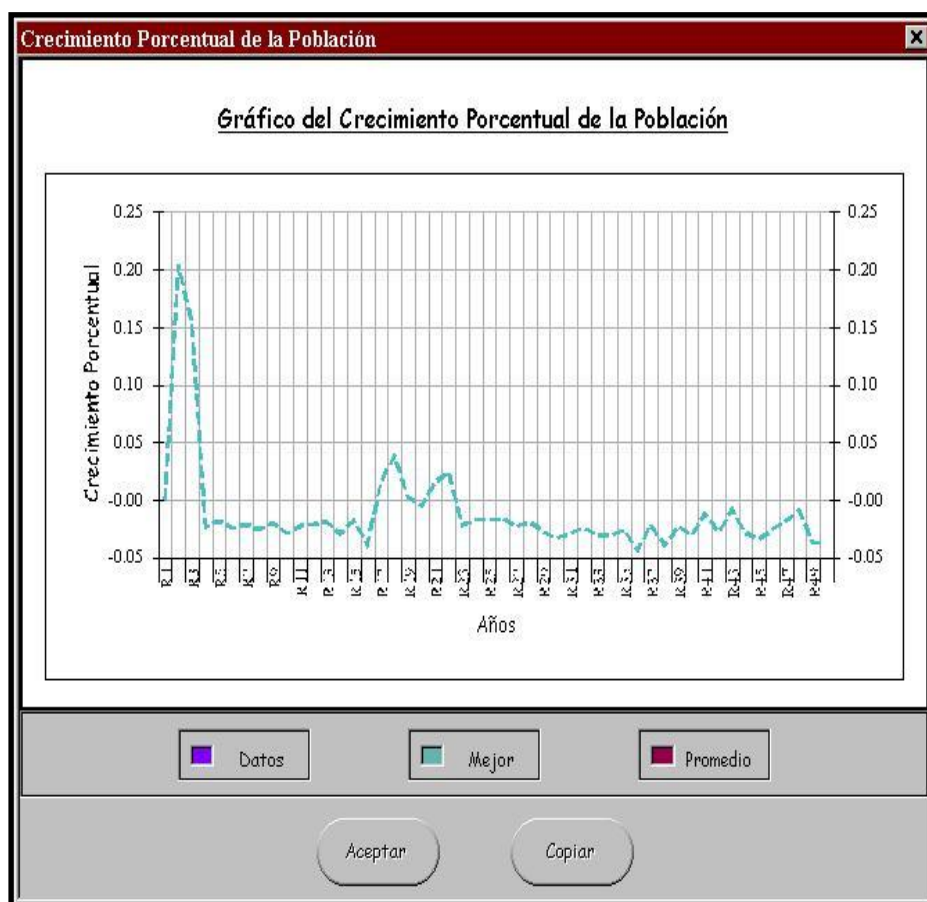


Gráfico de las aptitudes de los individuos de la población a través del tiempo

Figura 4.6

La tercera gráfica que puede elegir en esta aplicación es la de las variaciones porcentuales de crecimiento y decrecimiento como se

puede ver en la figura 4.7. En esta gráfica se observa en comparación con el año anterior de cuanto ha crecido o decrecido la población de manera porcentual. Usted tiene dos botones al igual que en las gráficas anteriores para copiar la imagen al portapapeles o para salir de esta pantalla.



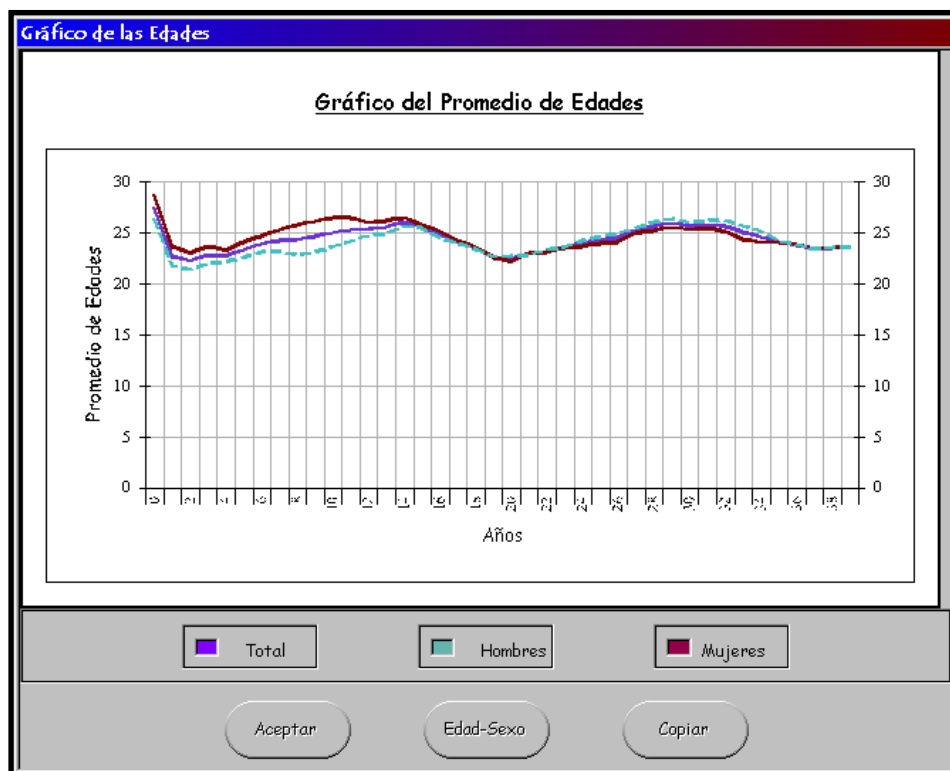
Pantalla del Gráfico de Crecimiento y decrecimiento porcentual a través del tiempo del Simulador del Algoritmos Genéticos – Modelo Poblacional

Figura 4.7

En realidad esta primera parte es un simulador del crecimiento de una población con un grupo de parámetros ingresados. En cada opción se utilizan valores completamente aleatorios, por lo que los resultados pueden variar notablemente de una corrida a otra.

Si usted ingresara a la parte de la aplicación que estima los betas del modelo exponencial y del modelo logístico, debe llenar cada uno de los campos, o elegir las opciones correspondientes, esta aplicación pide un intervalo de búsqueda para los valores de β_0 y β_1 , que corresponden a los parámetros del modelo exponencial o del modelo logístico. Mientras más pequeño sea el intervalo de búsqueda, se tendrá mejores resultados y una convergencia más rápida. Lo mismo ocurrirá si se aumenta el número de nodos o tamaño de la búsqueda.

Para utilizar el modelo exponencial, el tope de la búsqueda será cero (0), mientras que para utilizar el modelo logístico, deberá ingresar como límite superior del modelo un número mayor que todos los valores de los datos del archivo de datos.

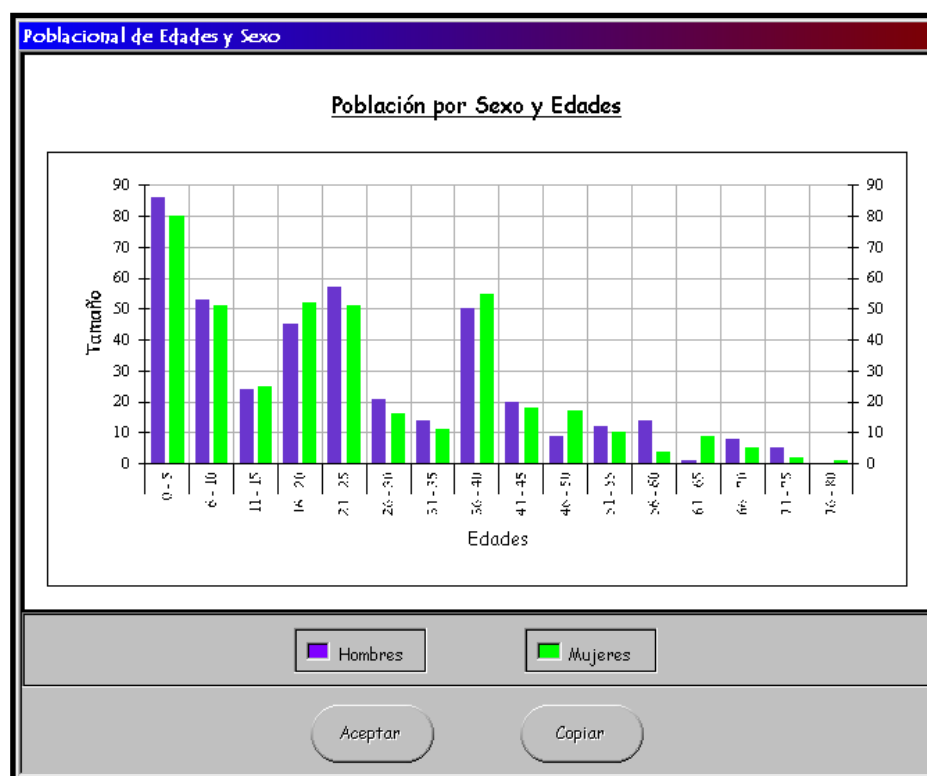


Pantalla del Gráfico de las Edades promedio de la población a través del tiempo del Simulador del Algoritmos Genéticos – Modelo Poblacional

Figura 4.8

En esta pantalla, además de encontrar los dos botones que encuentra en las pantallas anteriores, encontrará un botón que le permitirá ver la población en el último año dividida por edad y por sexo.

Además puede copiar al portapapeles el gráfico y salir de ésta con el botón de aceptar.



Pantalla del Gráfico de la población por Edades y por sexo en el último año de simulación del Algoritmos Genéticos – Modelo Poblacional

Figura 4.9

Si usted presiona el botón Edad-Sexo, aparecerá un gráfico como este, que muestra por edades y por sexo, los individuos de la población en el último año de la simulación.

The screenshot shows a software window titled "Modelos dependientes del Tiempo". It contains several sections for configuring a genetic algorithm:

- Parámetros de la población:**
 - B0: Input field with value 0, and a button with value 1.
 - B1: Input field with value 0, and a button with value 1.
 - Límite Superior de la Población: Input field with value 0.
- Algoritmos Genéticos:**
 - Número de iteraciones: Input field with value 20.
 - Prob. Crossover: Input field with value 0.9.
 - Prob. Mutación: Input field with value 0.4.
 - Tamaño de la Búsqueda: Input field with value 64.
- Selección:**
 - ☒ Torneo
 - ☐ Ruleta
- Adatación:**
 - ☐ Propagar el mejor
 - ☐ Usar Elitismo

On the right side of the window, there is a display showing the binary string "1010" in blue, and three buttons labeled "Abrir", "Crossover", and "Mutación". At the bottom right, there is a graphic of a DNA double helix. At the bottom left, there are two buttons labeled "Ejecutar" and "Salir".

Pantalla Principal del Modelo dependiente del tiempo del Algoritmos Genéticos – Modelo Poblacional

Figura 4.10

En esta parte de la aplicación, debe elegir el botón ejecutar para comenzar la ejecución de la aplicación; pero antes debe haberse ingresado correctamente cada uno de los parámetros que pide la

aplicación. Estos parámetros deben ser ingresados bajo las mismas condiciones que en el simulador del Algoritmos Genéticos – Modelo Poblacional. Finalmente, luego de ejecutarse le mostrará los resultados para los β_0 y β_1 . Mostrando dos resultados, que son los mejores estimadores y los estimadores promedios.

The screenshot shows a software window titled "Resultados" with a red title bar. It contains two main columns of results and a central DNA model visualization.

AG - Mejores Estimadores:		Mínimos cuadrados:	
B0:	9355.46875	B0:	9865.29104554
B1:	0.015625	B1:	0.01893850368
SCE:	0.93632454670	SCE:	0.159818776994
MCE:	0.02035488145	MCE:	0.00347432123
F*:	124.173354768	F*:	950.988540785
R^2:	0.787207817312	R^2:	0.95589596797

Below the results, there is a section labeled "Modelo:" containing two text boxes with mathematical formulas:

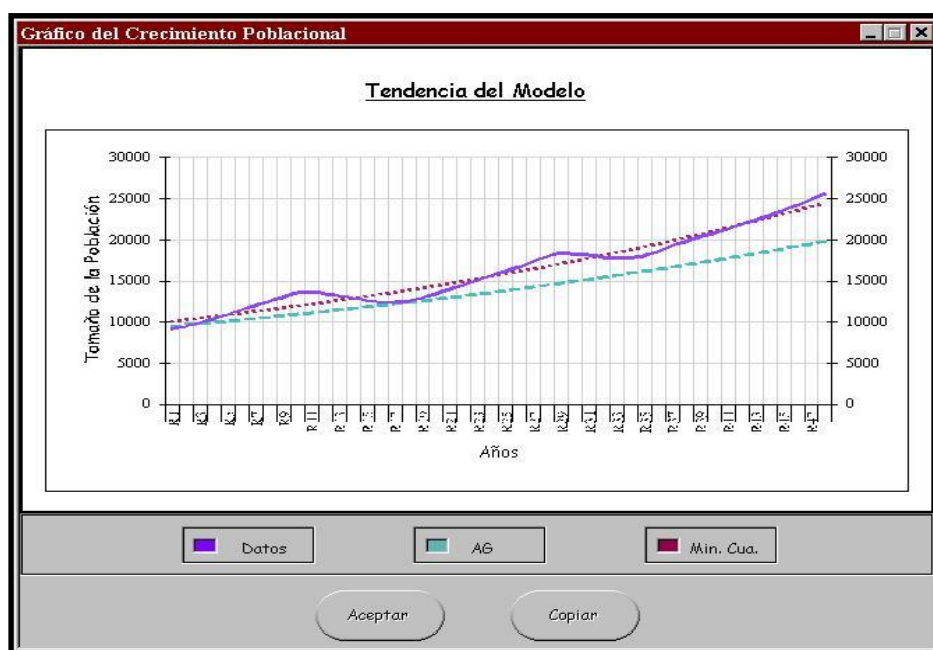
- AG - Mejores Estimadores: $P(t) = 1 / (-1.08e-062 + 9355 * 1.563e-002^t)$
- Est. Mínimos Cuadrados: $P(t) = 1 / (-1.08e-062 + 9865 * 1.894e-002^t)$

At the bottom of the window, there are four buttons: "Aceptar", "Modelo", "Error", and "Intervalos".

**Pantalla de resultados para los modelos dependientes del tiempo
mediante algoritmos genéticos
Figura 4.11**

Además usted podrá ver una gráfica de la tendencia del modelo poblacional que ha resultado de la ejecución del algoritmo genético, así como la gráfica de los errores y una pantalla que muestra los intervalos de confianza.

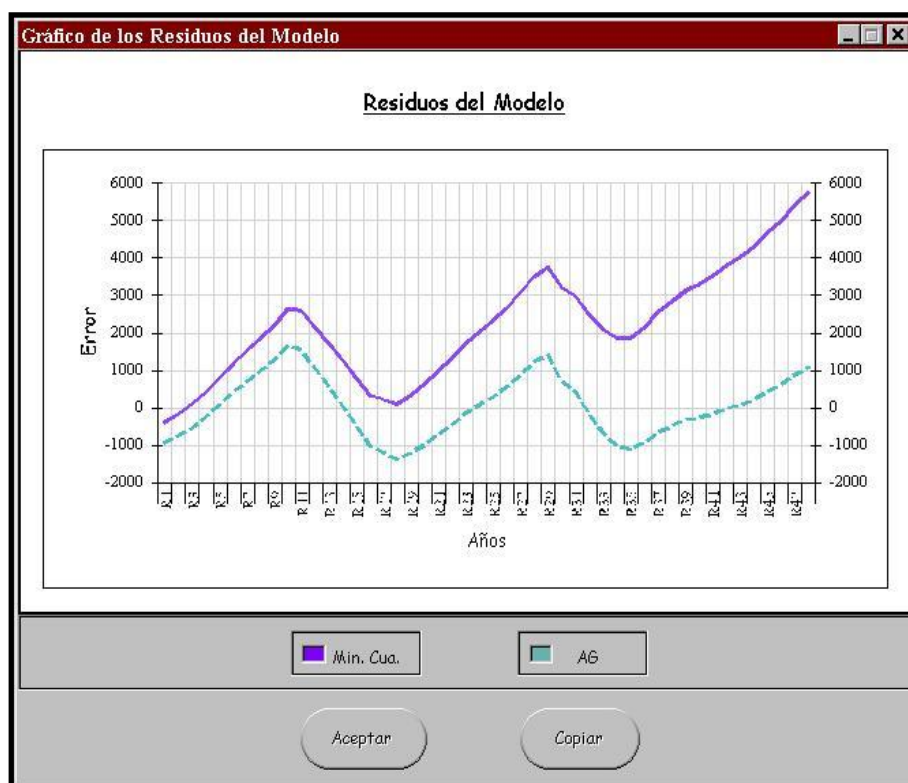
El gráfico de la tendencia exponencial o logística es igual a las otras gráficas, ya que podrá copiar la imagen al portapapeles haciendo un pulso con el ratón sobre el botón adecuado. La pantalla le mostrará tres líneas que representan los datos leídos por la aplicación, el modelo de los mejores estimadores con el algoritmo genético, y el modelo de los mínimos cuadrados.



Pantalla del Gráfico de los estimadores del modelo de crecimiento exponencial o logístico dependientes del tiempo

Figura 4.12

Otra opción es la gráfica de errores, la cual presentará dos líneas que representan el error del método de los mínimos cuadrados, y el error de los algoritmos genéticos en sus mejores estimadores.



Pantalla del Gráfico de los errores de los modelos de crecimiento exponencial o logístico pero dependientes del tiempo.

Figura 4.13

Finalmente puede escoger la opción de intervalos de confianza para tener una mejor visión de sus parámetros dentro de un intervalo de confianza que va a cambiar de acuerdo a la confianza que usted escoja. En la parte superior de la pantalla puede observar los intervalos de confianza para los estimadores del modelo con algoritmos genéticos, mientras que en la parte

izquierda puede observar los estimadores mediante el método de mínimos cuadrados. Esta pantalla también le mostrará el valor de $t_{\alpha/2}$ con el que se trabaja para dicho intervalo.

AG - Mejores Estimadores			
	Inferior	Superior	Prueba t
B0:	9865.20904428	9865.37304771	235799.582425
B1:	0.01602550243	0.02185249756	12.74085154344

Mínimos Cuadrados			
	Inferior	Superior	Prueba t
B0:	9355.43487320	9355.50262679	541276.694246
B1:	0.01442136720	0.01682863279	25.4438064705

$t(\alpha/2):$

Confianza

☐ 99 %

☐ 98 %

☒ 95 %

☐ 90 %

☐ 80 %

Aceptar

Pantalla de los intervalos de confianza de los modelos de crecimiento exponencial o logístico que dependen del tiempo.

Figura 4.14

Ambos modelos anteriores, exponencial y logístico, dependen única y exclusivamente del tiempo; pero también se puede buscar dentro de un grupo de posibles variables mediante los algoritmos genéticos una modelo que tenga mejor explicación, dependiendo de varias variables.

Pantalla Principal del Modelo dependiente de varias variables del Algoritmos Genéticos – Modelo Poblacional

Figura 4.15

En esta parte de la aplicación, debe elegir el botón ejecutar para comenzar la ejecución de la aplicación; pero antes debe haberse ingresado correctamente cada uno de los parámetros que pide la aplicación. Estos parámetros deben ser ingresados bajo las mismas condiciones que en el simulador del Algoritmos Genéticos – Modelo Poblacional. Finalmente, luego de ejecutarse le mostrará los resultados para los estimadores de los betas. Mostrando dos resultados, que son los mejores estimadores y los estimadores que pidió el usuario.

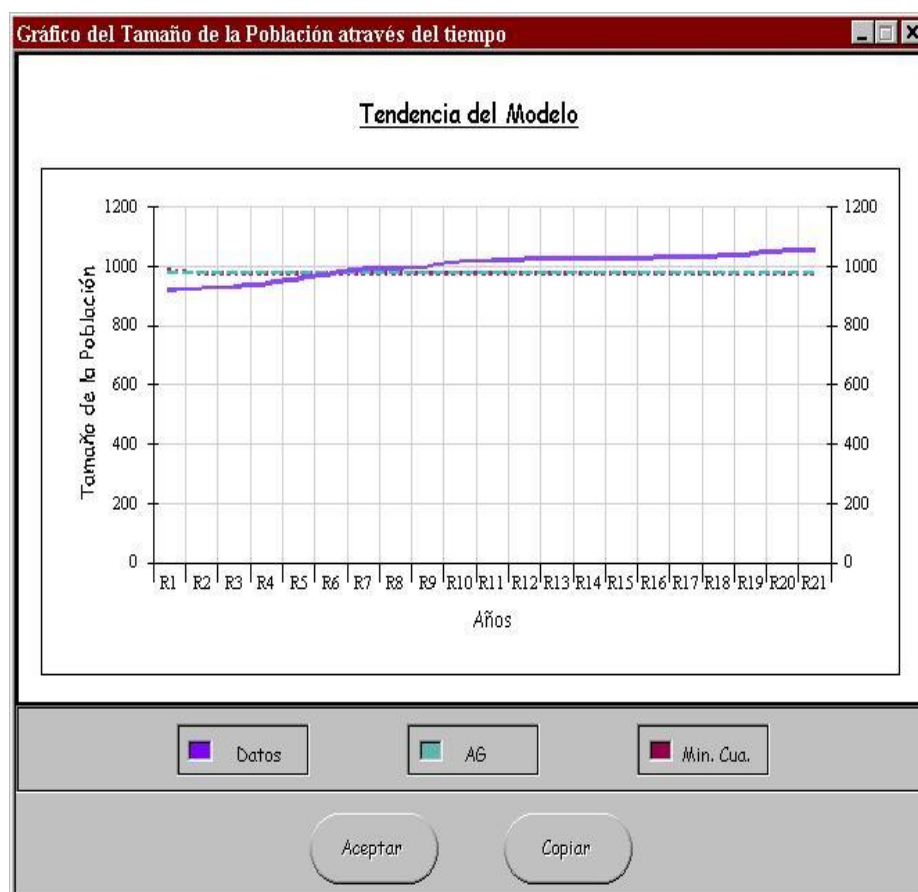
AG - Mejores Modelo		Mínimos cuadrados:	
B0:	977.23683	B0:	975.583455
B1:	0	B1:	0.000381
B2:	-0.000785	B2:	0
B3:	0.000627	B3:	0
B4:	0.001836	B4:	0.001641
SCE:	0.03247422952	SCE:	0.03287996706
MCE:	0.00202963934	MCE:	0.001934115709
F*:	5.767170112447	F*:	9.07800550598
R^2:	0.519541310951	R^2:	0.516441157268

Buttons: Aceptar, Tamaño, Error

Pantalla de resultados para los modelos dependientes de varias variables y mediante algoritmos genéticos

Figura 4.16

Esta pantalla cuenta con tres botones, que mostrarán el gráfico de los residuos, el gráfico de la tendencia del modelo, a través del tiempo, y el botón aceptar para salir.

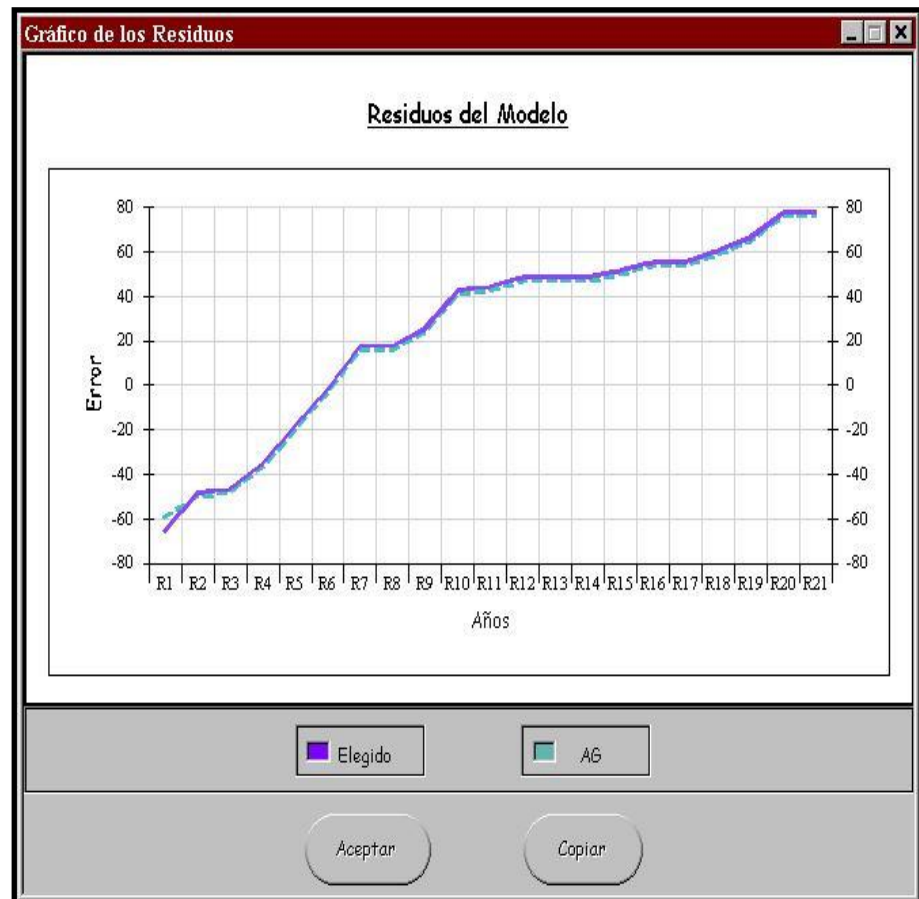


Pantalla del Gráfico de la tendencia del modelo de crecimiento exponencial o logístico dependientes de varias variables

Figura 4.17

En esta parte usted podrá copiar la imagen al portapapeles con el botón copiar o puede salir con el botón aceptar.

Otra opción que se tiene es el gráfico de los errores o residuos del modelo.



Pantalla del Gráfico de los errores de los modelos de crecimiento exponencial o logístico pero dependientes varias variables.

Figura 4.18

Además, en cualquier tipo de modelo, ya sea dependiente del tiempo o de las otras variables, usted deberá abrir un archivo de datos, antes de realizar la ejecución correspondiente del algoritmo genético de optimización.



Pantalla para abrir un archivo de datos

Figura 4.19

En esta pantalla el usuario debe ingresar el nombre del archivo que contiene los datos correspondientes. Este archivo será de texto. Luego debe hacer un pulso con el ratón en el botón aceptar. Si lo encontrase se mostrará un mensaje indicándolo, o lo contrario.



Mensaje de búsqueda del archivo de datos

Figura 4.20

Otra opción con la que cuenta cualquiera de las tres partes de la aplicación, es la configuración avanzada tanto para la mutación, como del crossover.

Si usted elige el crossover de tipo avanzado, aparecerá una pantalla similar a la que muestra la figura 4.19, y en ella podrá elegir el tipo de crossover y el lugar donde realizará el crossover. Para salir de esta pantalla deberá dar un pulso con el ratón sobre el botón aceptar para guardar la configuración, o un pulso en el botón cancelar para salir sin realizar cambio alguno.



Pantalla de la configuración avanzada para el crossover

Figura 4.21

Por último, en la pantalla de mutación avanzada, usted también puede elegir el tipo de mutación que utilizará, así como cuando realizar la mutación. Para salir de esta pantalla deberá dar un pulso con el ratón sobre el botón aceptar para guardar la configuración, o un pulso en el botón cancelar para salir sin realizar cambio alguno.



Pantalla de la configuración avanzada para el crossover

Figura 4.22

4.3. Manual técnico

4.3.1. Diagramas a nivel de contexto

Diagrama a nivel de contexto del algoritmo genético de simulación.

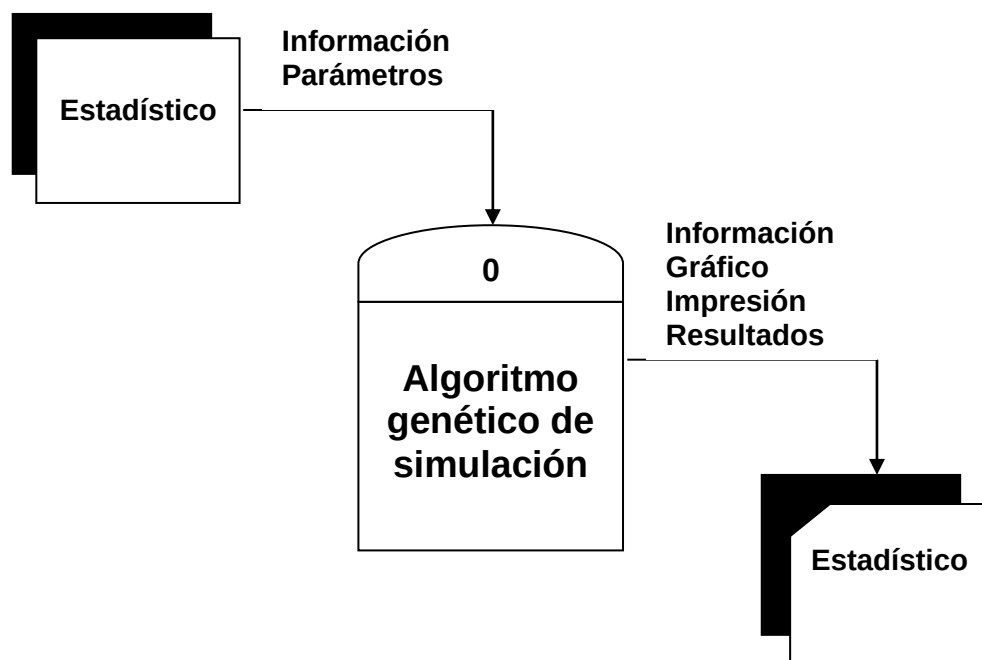


Diagrama de contexto del algoritmo genético de simulación

Figura 4.23

**Diagrama a nivel de contexto del algoritmo genético para el
modelo exponencial o logístico dependientes del tiempo**

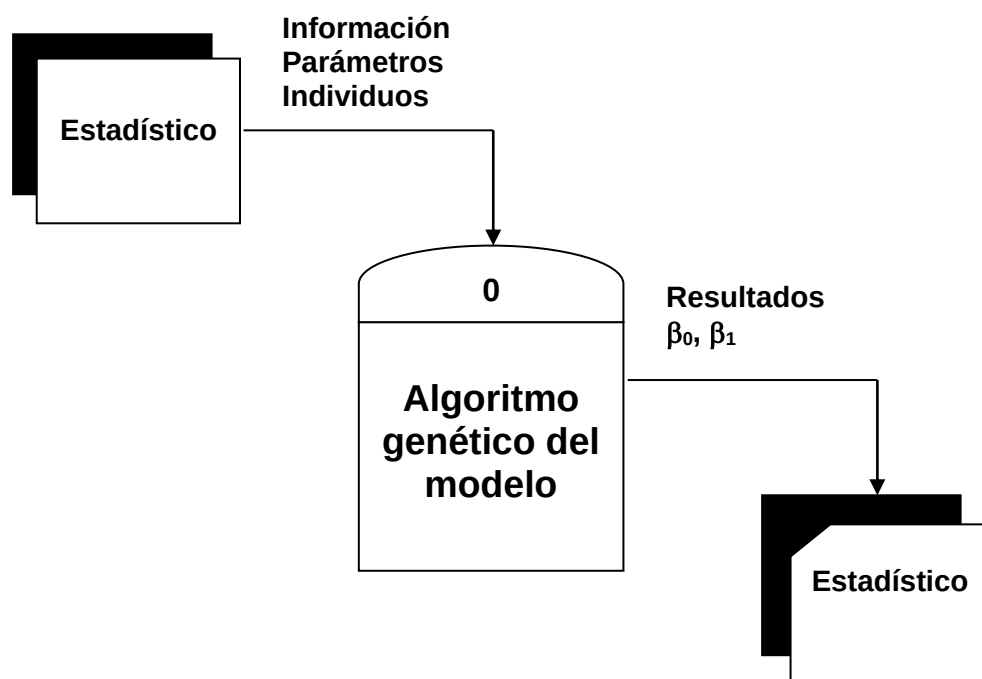


Diagrama de contexto del algoritmo genético del modelo dependiente del tiempo

Figura 4.24

Diagrama a nivel de contexto del algoritmo genético para el modelo exponencial o logístico dependiente de varias variables.

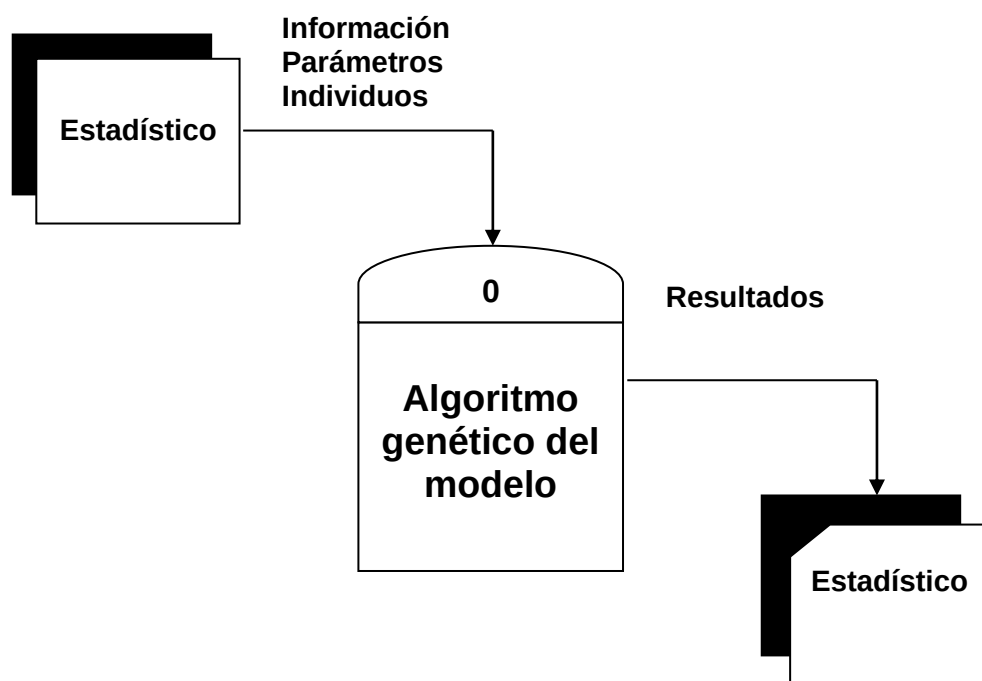


Diagrama de contexto del algoritmo genético del modelo dependiente de varias variables

Figura 4.25

4.3.2. Diagramas de nivel 0

Diagramas de nivel 0 del algoritmo genético de simulación

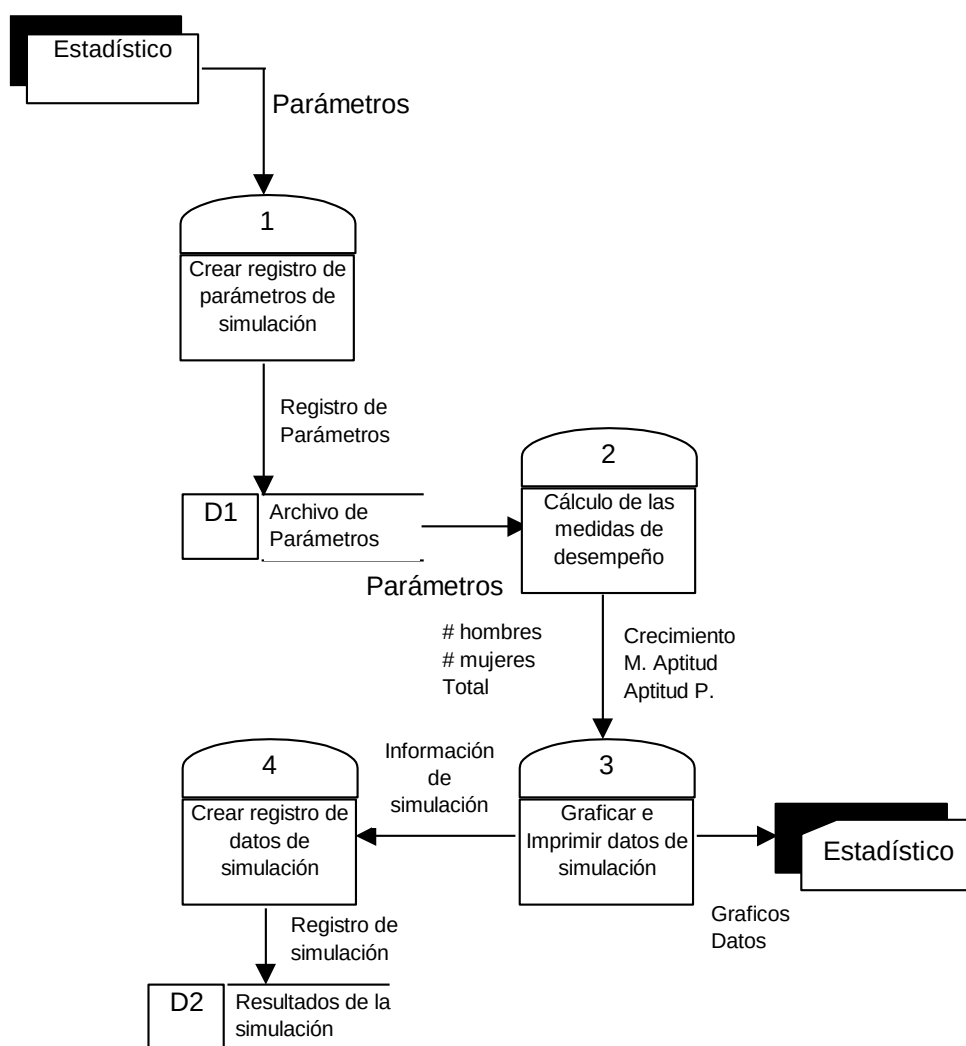


Diagrama cero del algoritmo genético de simulación

Figura 4.26

Diagramas de nivel 0 del algoritmo genético del modelo dependiente del tiempo

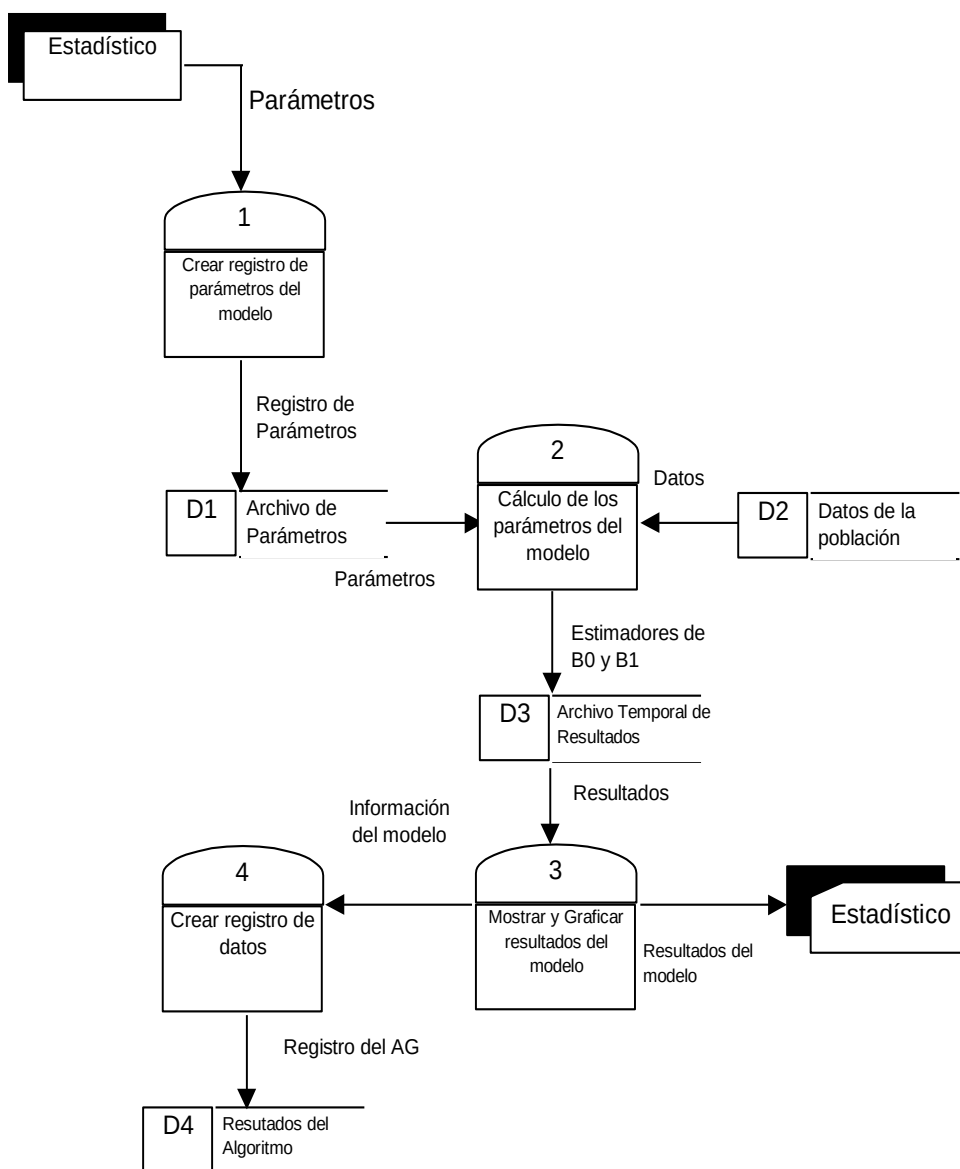


Diagrama cero del algoritmo genético de los modelos dependientes del tiempo

Figura 4.27

Diagramas de nivel 1 del proceso de cálculo del algoritmo genético de los modelos dependientes del tiempo.

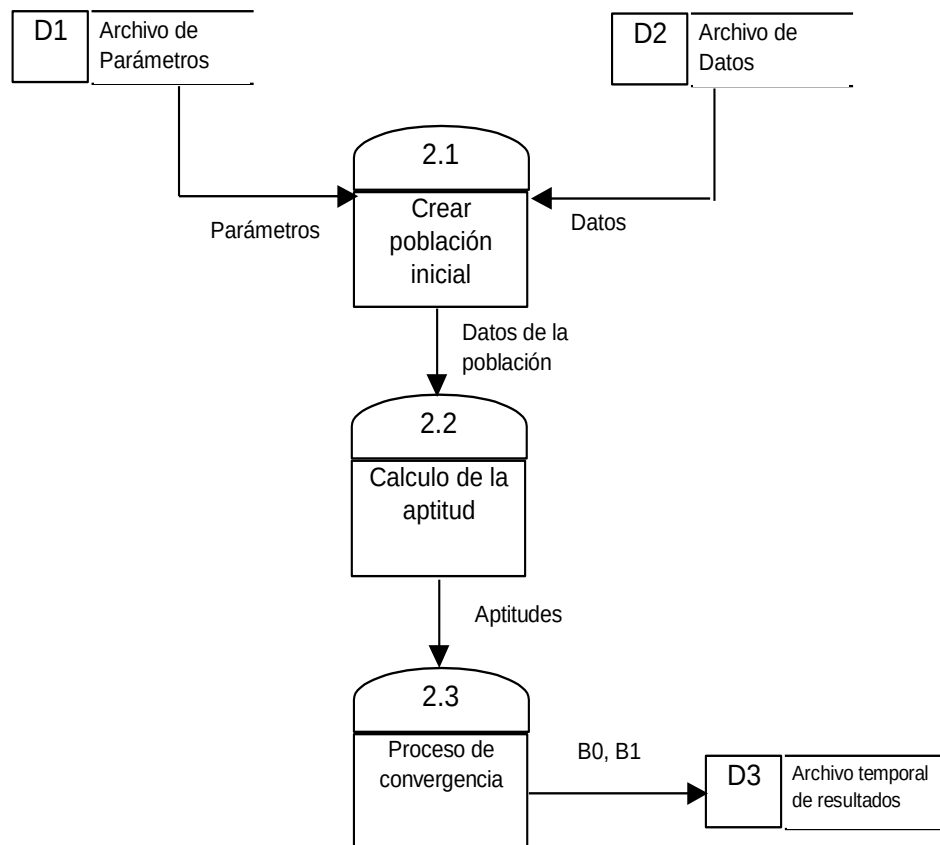


Diagrama de nivel 1 del proceso de cálculo del algoritmo genético de los modelos dependientes del tiempo para los beta

Figura 4.28

Diagramas de nivel 0 del algoritmo genético de los modelos dependientes de varias variables.

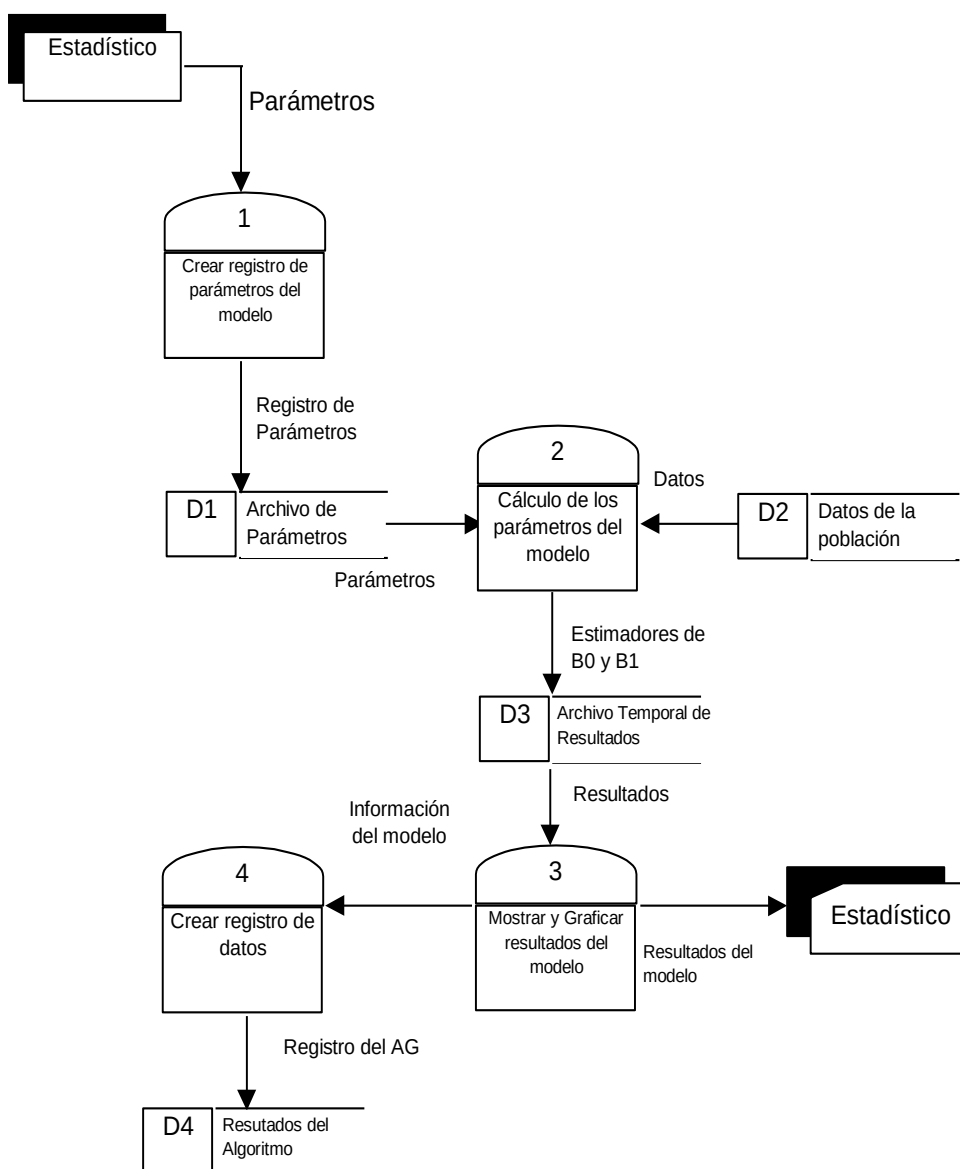


Diagrama cero del algoritmo genético de los modelos dependientes de varias variables

Figura 4.29

Diagramas de nivel 1 del calculo del algoritmo genético del modelo de selección.

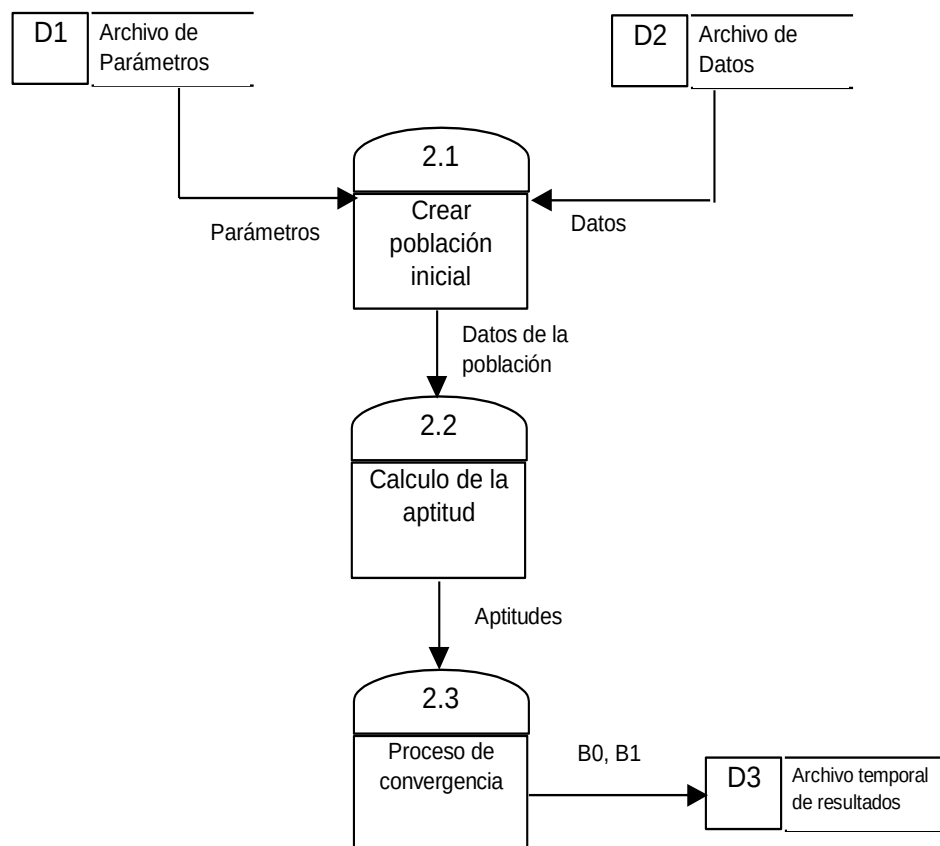


Diagrama de nivel 1 del proceso de cálculo del algoritmo genético de los modelos dependientes de varias variables para los beta

Figura 4.30

4.3.3. Descripción de las variables y procesos de los algoritmos genéticos

4.3.3.1. Descripción de las variables y procesos del algoritmo genético de simulación

Descripción de las variables globales

<i>longicromo</i>	Longitud de los cromosomas
<i>int npob</i>	Tamaño de la población
<i>int genera</i>	Número de generaciones
<i>Double Pcro</i>	Probabilidad de crossover
<i>Double Pmu;</i>	Probabilidad de mutar
<i>int sele</i>	Tipo de selección
<i>int promejor</i>	Propagar el mejor?
<i>int limites</i>	Crossover solo en los limites?
<i>int Elitismo</i>	Usar elitismo?
<i>int tipocros</i>	Tipo de crossover
<i>int tipomuta</i>	Tipo de mutadcion
<i>int menor</i>	Mutar sólo los menores
<i>int exigir</i>	Exigir mutacion
<i>long iteracion</i>	Contador de las iteraciones
<i>double mayaptitud</i>	Mayor aptitud
<i>double proaptitud</i>	Aptitud promedio

<i>double proedad</i>	Edad promedio
<i>double suma</i>	Suma de las aptitudes
<i>double nhombres</i>	Numero de hombres
<i>double nmujeres</i>	Numero de mujeres
<i>double nanterior</i>	Numero anterior de la poblacion
<i>double logi_apti</i>	Poblacion para aptitud minima
<i>double nacea</i>	Límite inferior de las edades en las que se reproducen
<i>double naceb</i>	Límite superior de las edades en las que se reproducen
<i>double muere</i>	Edad en la que muere
<i>double nhijos</i>	Promedio de hijos
<i>double crecimiento</i>	Crecimiento porcentual año a año

Descripción de las estructuras utilizadas

```
struct genetico {
char cromosoma[longicromo+1];
char cromosoma2[longicromo+1];
double geno;
double aptitud;
double dispo;
};
```

Estructura de tipo genético

Cuadro 4.1

Estructura de caracter genetico

Contiene un individuo de tipo cadena de caracteres de longitud longicromo+1, la cual se le denomina cromosoma.

Contiene un individuo seleccionado del mismo tipo que el individuo de trabajo, se denomina cromosoma2 y se utiliza en la selección.

El genotipo en la codificación a base decimal del cromosoma, y luego se hace la conversión a un intervalo. Esta es de tipo double.

La aptitud, es el genotipo evaluado en la función de aptitud, y es de tipo double.

La aptitud relativa es de tipo double, y es la aptitud en un intervalo de cero a uno.

```
struct datos_persona {
    int edad_actual;
    int edad_limite;
    char sexo;
    int reproa;
    int reprob;
    int num_hijos;
    int num_tot_hijos;
};
```

Estrctura de tipo datos de la persona

Cuadro 4.2

Estructura con datos personales

Contiene la edad actual del individuo

La edad de fallecimiento del individuo

El sexo del individuo, que puede ser masculino o femenino

El límite inferior del intervalo en que puede reproducirse un individuo

El límite superior del intervalo en que puede reproducirse un individuo

El número de hijos que actualmente tiene

El número total de hijos que tendrá dicho individuo

```
struct persona {
    genetico gene;
    datos_persona datos;
};
```

Estructura de tipo persona

Cuadro 4.3

Estructura de persona

Parte reproductiva del individuo, que posee todas las cualidades del algoritmo genético.

Parte de los datos personales del individuo.

```
struct poblacion {
    persona info;
    poblacion *sig;
}*Hpob,*Mpob;
```

Estructura de tipo población

Cuadro 4.4

Estructura de poblacion con personas

Contiene cada persona de la población enlazadas mediante un puntero al siguiente individuo y está separada entre hombres y mujeres.

```
struct selec {
    double acu;
    int pos;
    selec *sig;
};
```

Estructura de tipo selección

Cuadro 4.5

Estructura para el método de selección de la Rued de la Ruleta

Contiene la función acumulada de aptitudes de los individuos, la posición en la que será asignado el individuo y unPuntero al siguiente seleccionado.

Descripción de los procedimientos y funciones

<i>AGpob(void);</i>	Constructor del Agpob, algoritmo genético de simulación poblacional. Inicializa el número de iteraciones.
<i>double normal(double,double);</i>	Función para generar números aleatorios normales $N(\mu, \sigma^2)$.
<i>void lee_configuracion(void);</i>	Lee la configuración ingresada del algoritmo genético.
<i>void lee_parametros(void);</i>	Lee los parametros ingresados del modelo poblacional.
<i>void generainicial(void);</i>	Genera la poblacion inicial, incluyendoles un genotipo y una aptitud a cada uno de los individuos.
<i>double aptitud(double);</i>	Calcula la aptitud de cada individuo de la población.

<i>long random(long);</i>	Genera un número aleatorio uniforme $U(0,b)$.
<i>int muerte(double);</i>	Genera un número aleatorio con la distribución de edad límite de vida.
<i>double U01(void);</i>	Genera un número aleatorio de la distribución uniforme $U(0,1)$.
<i>int converge(void);</i>	Verifica si termina la ejecución del Algoritmo genético.
<i>int puede_reproducirse(poblacion *);</i>	Verifica si un individuo puede reproducirse.
<i>long num_reproduce(int);</i>	Halla el número de individuos que pueden reproducirse.
<i>void halla_aptitud01(void);</i>	Halla las aptitudes entre 0 y 1.
<i>void seleccion_Torneo(long,long);</i>	Selecciona los padres por medio del método del Torneo.

<i>void seleccion_Ruleta(int,long);</i>	Selecciona los padres por medio del método de la Rueda de la Ruleta.
<i>void crossover1(void);</i>	Realiza el crossover en un punto.
<i>void crossoverU(void);</i>	Realiza el crossover Uniforme.
<i>void mutacion1(int);</i>	Realiza la mutacion en un punto.
<i>void mutacionU(int);</i>	Realiza la mutacion Uniforme.
<i>void pasa_tiempo(void);</i>	Función que envejece los individuos de la población.
<i>void elimina_viejos(void);</i>	Elimina los individuos que pasan el limite de edad.
<i>double genotipo(char []);</i>	Genotipo del cromosoma.
<i>void reemplazo(void);</i>	Reemplazo la antigua poblacion.
<i>void halla_mayproaptitud(void);</i>	Halla la aptitud promedio y la mayor.
<i>void guardar_resultados(int);</i>	Guarda los resultados de una iteracion.

<i>void crear(void);</i>	Crea una poblacion con individuos.
<i>void destruir(void);</i>	Destruye una poblacion.
<i>void quitar_primeroh(void);</i>	Elimina el primero de la poblacion.
<i>void quitar_primerom(void);</i>	Elimina el primero de la poblacion.
<i>void agregar_hombre(persona);</i>	Agrega una persona.
<i>void agregar_mujer(persona);</i>	Agrega una persona.
<i>void ordenar(int,int);</i>	Ordena la poblacion.
<i>char conjuncion(char,char);</i>	Operador de conjunción.
<i>char disyuncion(char,char);</i>	Operador de disyunción.
<i>char no(char);</i>	Operador de negacion.
<i>char igual(char,char);</i>	Operador de igualdad.
<i>char diferente(char,char);</i>	Operador de diferencia.
<i>int _sele(void);</i>	Retorna el método de selección.
<i>int _tipomuta(void);</i>	Retorna el tipo de mutación.
<i>int _tipocros(void);</i>	Retorna el tipo de crossover.

<i>int _genera(void);</i>	Retorna el número de generaciones.
<i>int Logistica_permite(void);</i>	verifica si puede seguir creciendo.

4.3.3.2. Descripción de las variables y procesos del algoritmo genético de Optimización para los modelos dependientes del tiempo

Descripción de las variables globales

<i>MAXLON</i>	Número maximo de la longitud de los cromosomas
<i>double Y[3]</i>	Vector que contiene sumas propias de la función de adaptación
<i>double SCR</i>	Suma cuadratica de refresión
<i>double R2</i>	Coeficiente de Determinación
<i>long n</i>	Número de datos
<i>long iter</i>	Iteraciones
<i>long tambus</i>	Tamaño de búsqueda paralela
<i>double pcross</i>	Probabilidad de crossover
<i>double pmuta</i>	Probabilidad de mutacion
<i>tope</i>	Límite superior del crecimiento

	poblacional
<i>double b0a</i>	Límite inferior del intervalo de búsqueda del parámetro β_0
<i>double b0b</i>	Límite superior del intervalos de búsqueda del parámetro β_0
<i>double b1a</i>	Límite inferior del intervalos de búsqueda del parámetro β_1
<i>double b1b</i>	Límite superior del Intervalo de búsqueda del parametro β_1
<i>int tcross</i>	Tipo de crosover
<i>int tmuta</i>	Tipo de mutación
<i>int sele</i>	Tipo de selección
<i>BOOL extremos</i>	Utilizar crossover únicamente en los extremos
<i>BOOL menor</i>	Utilizar mutación únicamente en los de menor aptitud
<i>BOOL mejor</i>	Promover el mejor individuo
<i>BOOL elitismo</i>	Utilizar elitismo
<i>int longitud</i>	Longitud de los cromosomas
<i>double mayor</i>	Mayor aptitud
<i>double prome</i>	Aptitud promedio

double mayor_a β_0 de la mayor aptitud

double mayor_b β_1 de la mayor aptitud

Descripción de las estructuras utilizadas

```
struct persona {
    int cromos1[MAXLON+1];
    int cromos2[MAXLON+1];
    double ada;
    double dis;
    double a;
    double b;
};
```

Estructura de tipo persona

Cuadro 4.6

Estructura de tipo persona, nodos de búsqueda

Contiene al individuo o nodo que lo describo como cromos1, y el individuo seleccionado para la reproducción en el crossover, que lo identifico como cromos2. La adaptación del individuo, que en este caso es la suma de cuadrados del error, y la disposición del individuo, que es la función acumulada de las aptitudes. Además del β_0 del individuo y el β_1 del individuo.

```

struct pobla {
    persona individuo;
    pobla *sig;
} *pob;

```

Estructura de tipo población

Cuadro 4.7

Estructura de tipo población

Contiene al individuo que pertenece a la población y un puntero al siguiente individuo de la población. Esta lista contendrá n individuos y no va a variar su tamaño durante toda la ejecución.

Descripción de los procedimientos y funciones del algoritmo genético de optimización para los parámetros del modelo dependiente del tiempo.

<i>void sacar(void)</i>	Elimina el primero de la lista.
<i>void ruleta(void)</i>	Método de la ruleta
<i>void torneo(void)</i>	Método del torneo
<i>void crossover1(double,BOOL)</i>	Realiza el crossover en un punto
<i>void crossoverU(double)</i>	Realiza el crossover de tipo uniforme

<i>void mutacion1(double,int)</i>	Realiza la mutación en un punto
<i>void mutacionU(double,int)</i>	Realiza la mutación de tipo uniforme
<i>double genotipo(int [])</i>	Base 10 del cromosoma
<i>double adaptacion(double,double);</i>	Funcion de adaptacion de los individuos
<i>Double genotipo1(int A[]);</i>	Genotipo de β_0
<i>Double genotipo2(int A[]);</i>	Genotipo de β_1
<i>long random(long x);</i>	Número aleatorio uniforme discreto
<i>long preferencia_menor(double w);</i>	Número aleatorio con preferencia en los menores
<i>void crear(void);</i>	Crea la lista para los individuos iniciales
<i>void destruir(void);</i>	Destruye la lista
<i>void lee_configuracion(void);</i>	Lee la configuración ingresada
<i>void inicial(void);</i>	Genera la población inicial

<i>void halla_Sumas(void);</i>	Halla las sumas para la función de aptitud
<i>void ordenar(void);</i>	Ordena la lista de menor a mayor
<i>void crossover(void);</i>	Realiza el crossover
<i>void mutacion(void);</i>	Realiza la mutacion
<i>void seleccion(void);</i>	Realiza la selecció
<i>void reemplazo(void);</i>	Reemplaza la poblacion anterior con la nueva
<i>void maypro(void);</i>	Halla el mayor y el promedio
<i>double halla_sce(void);</i>	Halla la suma de cuadrados del error;
<i>void halla_resultados(void);</i>	Halla el Coeficiente de determinación
<i>BOOL converge(long);</i>	Verifica si la población converge
<i>double _a1(void);</i>	Retorna el estimador de β_0
<i>double _b1(void);</i>	Retorna el estimador de β_1

<i>double _SCE1(void);</i>	Retorna la suma de cuadrados del error
<i>long _iter(void);</i>	Retorna el número de iteraciones
<i>double _SCR(void);</i>	Retorna la suma de cuadrados de regresión
<i>double _R2(void);</i>	Retorna el coeficiente de determinación

4.3.3.3. Descripción de las variables y procesos del algoritmo genético de Optimización para los modelos dependientes de varias variables.

Descripción de las variables globales

<i>double SCE;</i>	Suma de cuadrados del error
<i>double MCE;</i>	Media de cuadrados del error
<i>double R2;</i>	Coeficiente de Determinación
<i>double Femp;</i>	Valor de la prueba F de Fisher
<i>long n;</i>	Número de datos
<i>double tope;</i>	valor maximo de la funcion
<i>long iter;</i>	Iteraciones
<i>long tambus;</i>	tamaño de búsqueda paralela
<i>double pcross;</i>	Probabilidad de Crossover

<i>double pmuta;</i>	Probabilidad de Mutacion
<i>int tcross;</i>	Tipo de crossover
<i>int tmuta;</i>	Tipo de mutación
<i>int sele;</i>	Tipo de selección
<i>BOOL extremos;</i>	Crossover en los extremos
<i>BOOL menor;</i>	Mutar los menores
<i>BOOL mejor;</i>	Propagar el mejor
<i>BOOL elitismo;</i>	Utilizar elitismo
<i>int longitud;</i>	Longitud de los cromosomas
<i>double mayor;</i>	Mayor aptitud
<i>BOOL mayor_cro[MAXLON];</i>	Cromosoma de mayor aptitud
<i>double prome;</i>	Aptitud promedio

Descripción de las estructuras utilizadas

```
struct persona {
    BOOL cromosoma[MAXLON];
    double ada;
    double dis;
};
```

Estructura de tipo persona

Cuadro 4.8

Estructura de tipo persona, nodos de búsqueda

Contiene al individuo o nodo que lo describo como *cromo1*, y el individuo seleccionado para la reproducción en el crossover, que lo identifico como *cromo2*. La adaptación del individuo, que en este caso es la suma de cuadrados del error, y la disposición del individuo, que es la función acumulada de las aptitudes. Además del β_0 del individuo y el β_1 del individuo.

```
struct pobla {
    persona individuo;
    pobla *sig;
}*pob2;
```

Estructura de tipo población

Cuadro 4.9

Estructura de tipo población

Contiene al individuo que pertenece a la población y un puntero al siguiente individuo de la población. Esta lista contendrá *n* individuos y no va a variar su tamaño durante toda la ejecución.

Descripción de los procedimientos y funciones del algoritmo genético de optimización para los parámetros del modelo dependiente de varias variables.

<i>void sacar(void);</i>	Elimina el primero de la lista
<i>void ruleta(void);</i>	Método de la ruleta
<i>void torneo(void);</i>	Método del torneo
<i>void crossover1(double,BOOL);</i>	Crossover en 1 punto
<i>void crossoverU(double);</i>	Crossover Uniforme
<i>void mutacion1(double,int);</i>	Mutación en 1 punto
<i>void mutacionU(double,int);</i>	Mutación Uniforme
<i>double adaptacion(BOOL A[]);</i>	Funcion de adaptacion
<i>long random(long x);</i>	Número aleatorio discreto en tre 0 y x
<i>long preferencia_menor(double w);</i>	Número aleatorio con preferencia en los menores
<i>void crear(void);</i>	Crea la lista
<i>void destruir(void);</i>	Destruye la lista
<i>void lee_configuracion(void);</i>	Lee la configuración ingresada por el usuario

<i>void inicial(void);</i>	Genera la población inicial
<i>void halla_Sumas(void);</i>	Halla las sumas necesarias para la regresión
<i>void ordenar(void);</i>	Ordena la población de mayor a menor aptitud
<i>void crossover(void);</i>	Realiza el crosover
<i>void mutacion(void);</i>	Realiza la mutación
<i>void seleccion(void);</i>	Realiza la selección
<i>void reemplazo(void);</i>	Realiza el reemplazo de la población
<i>void maypro(void);</i>	Halla el mayor y el promedio
<i>double halla_sce(void);</i>	Halla la suma de cuadrados del error
<i>void halla_resultados(void);</i>	Halla los resultados
<i>BOOL converge(long);</i>	Verifica si la población a convergido
<i>double _SCE1(void);</i>	Retorna la suma de cuadrados del error

double _MCE(void);

Retorna la media de
cuadrados del error

double _R2(void);

Retorna el coeficiente de
determinación

double _Femp(void);

Retorna el valor de la
prueba F de Fisher

long _iter(void);

Retorna el número de
iteraciones

CAPITULO V

5. Análisis de las variables

5.1. Introducción

En este capítulo voy a realizar predicciones de los resultados arrojados por la aplicación que he desarrollado, algoritmos genéticos – modelo poblacional, y a compararlos con los mismos. Estas predicciones las haré basándome en los modelos clásicos de crecimiento poblacional, recordando que el crecimiento poblacional está dado por la diferencia de los índices de natalidad y mortalidad.

5.2. Análisis de los resultados de la aplicación: Algoritmos genéticos – modelo poblacional.

Para todas las siguientes ejecuciones, trabajaré con la configuración del límite de la población $w=80$, y con un intervalo de fecundidad desde los 15 hasta los 44 años de edad, tanto en hombres como en mujeres, aunque luego haré variar estos parámetros. Utilizaré también 25 iteraciones, en las que debe crecer o decrecer la población. Mostraré entonces la tabla de resultados obtenidos, y el gráfico de los años en las abscisas, y en las ordenadas el tamaño de la población. Trabajaré con 100 hombres y 100 mujeres como tamaño inicial de la población, y observaré el comportamiento de la curva, haciendo variar el número de hijos.

Lo que nos interesa es realizar comparaciones entre los métodos de crossover en un punto y crossover uniforme, así como la mutación en un punto y la mutación uniforme, ya que estos son los operadores básicos del algoritmo genético básico. De igual manera, es interesante comparar los resultados obtenidos con una configuración con elitismo y otra sin elitismo, ya que el elitismo garantiza una mejor aptitud de los individuos en un menor número de iteraciones. Podría ser también al exigir o no exigir mutación,

ya que esto provoca una aptitud más aleatoria. Finalmente saber si varía utilizando el método de la rueda de la ruleta o el del torneo para llevar a cabo la selección. De este modo podremos observar el comportamiento de la función de aptitud, y por consiguiente encontraremos un crecimiento o decrecimiento de la población.

Inicialmente realizaré una configuración del modelo poblacional, ajustando los datos a una población que desea mantenerse estable o que decrezca, esto es, que tengan un número promedio de 2 hijos; pero posteriormente, cambiaré el número de hijos de dos a un cuatro, de tal manera que la población tenga un ligero crecimiento, con esto trataré de comprobar que la población aumenta al aumentar el número promedio de hijos, mientras que podría disminuir si reducimos el intervalo de edades de reproducción, ya que en este caso no damos oportunidad a las personas a tener el número deseado de hijos, o si disminuimos el w de la edad límite de vida, a que estaríamos hablando de una población que muere a una edad temprana.

Considerando dos hijos

En una población con solamente dos hijos, debe mantenerse el tamaño de la población, aunque en realidad crecerá y decrecerá hasta estabilizarse, de tal manera que los dos hijos reemplacen a

sus padres. Aunque las variaciones se darán debido a que este modelo algorítmico considera el sexo de los individuos como una variable importante.

Crossover

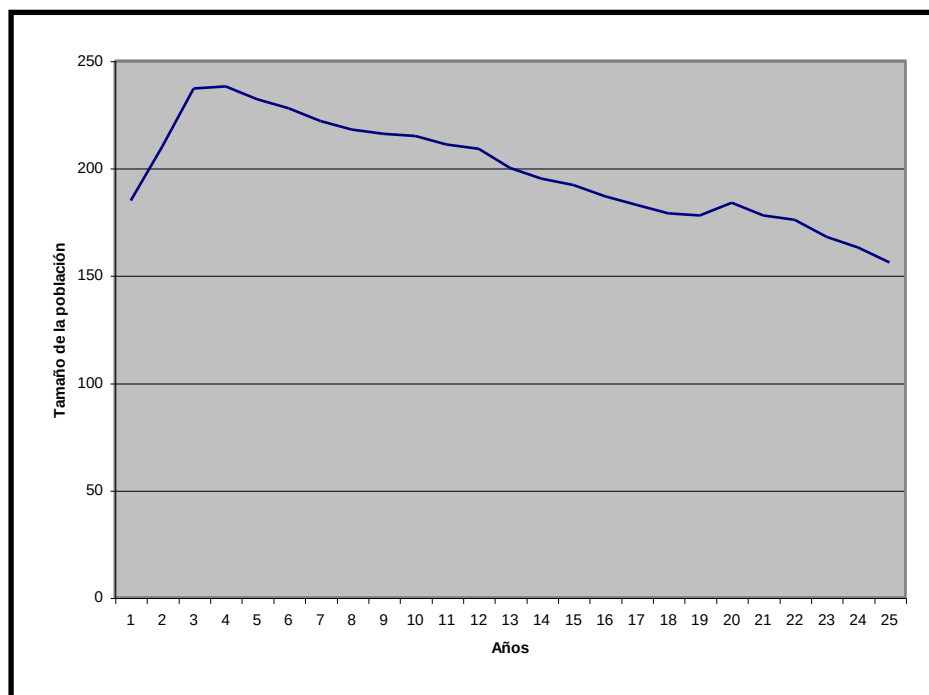
Primero realizaré dos ejecuciones comparando el crossover en un punto y el crossover uniforme, ambos con mutación en un solo punto, y con el modelo de selección de la rueda de la ruleta, y la primera tabla será comparada con las demás.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	90	95	185	+0.000000	0.058356	0.040578
2	103	107	210	+0.135135	0.054772	0.037193
3	115	122	237	+0.128571	0.051558	0.035018
4	116	122	238	+0.004219	0.051450	0.034972
5	115	117	232	-0.025210	0.052111	0.035281
6	113	115	228	-0.017241	0.052566	0.035391
7	111	111	222	-0.026316	0.053271	0.035625
8	110	108	218	-0.018018	0.053758	0.035825
9	109	107	216	-0.009174	0.054006	0.036091
10	108	107	215	-0.004630	0.054132	0.036153
11	106	105	211	-0.018605	0.054642	0.036421
12	105	104	209	-0.009479	0.054903	0.036435
13	100	100	200	-0.043062	0.056125	0.037119
14	99	96	195	-0.025000	0.056840	0.037295
15	98	94	192	-0.015385	0.057282	0.037557
16	97	90	187	-0.026042	0.058043	0.037929
17	97	86	183	-0.021390	0.058674	0.038002
18	94	85	179	-0.021858	0.059326	0.038433
19	93	85	178	-0.005587	0.059492	0.038702
20	95	89	184	+0.033708	0.058514	0.038122
21	90	88	178	-0.032609	0.059492	0.038389
22	90	86	176	-0.011236	0.059829	0.038334
23	87	81	168	-0.045455	0.061237	0.039388
24	83	80	163	-0.029762	0.062169	0.039743
25	80	76	156	-0.042945	0.063549	0.040872
Promedio	100.16	98.24	198.40	-0.005895	0.056644	0.037395

Tabla de la simulación con dos hijos, crossover en un punto, mutación en un punto, selección con el método de la rueda de la ruleta, sin elitismo, sin exigir mutación.

Tabla III

En esta ejecución se tuvo un decrecimiento promedio del 0,59% en 25 años, lo que es relativamente poco, y sería un decrecimiento bastante bueno en caso de querer disminuir la población.



Tamaño poblacional de la primera simulación con dos hijos

Figura 5.1

Entonces se cumple el supuesto de decrecimiento al utilizar únicamente dos hijos en promedio, y notamos que a medida que la población decrece, la aptitud promedio de la misma aumenta, es decir que aumentan los recursos para cada individuo de la población.

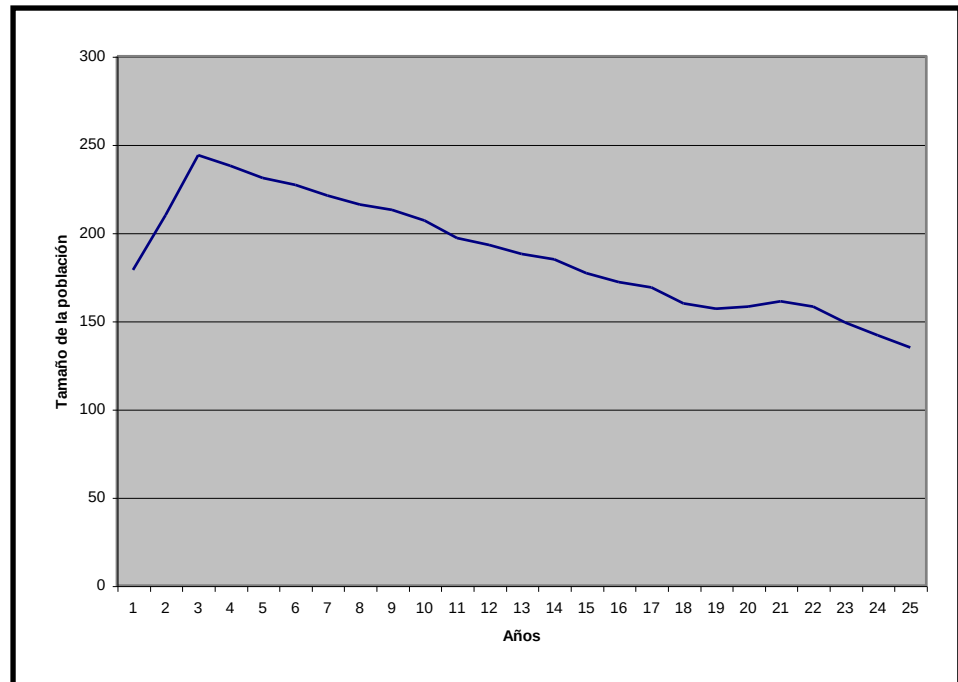
En la siguiente ejecución, utilicé crossover uniforme:

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	88	91	179	+0.000000	0.059326	0.039388
2	108	102	210	+0.173184	0.054772	0.036617
3	127	117	244	+0.161905	0.050813	0.033805
4	124	114	238	-0.024590	0.05145	0.034243
5	120	111	231	-0.029412	0.052223	0.035216
6	119	108	227	-0.017316	0.052681	0.035312
7	118	103	221	-0.026432	0.053392	0.036021
8	117	99	216	-0.022624	0.054006	0.036316
9	116	97	213	-0.013889	0.054385	0.036505
10	114	93	207	-0.028169	0.055168	0.036784
11	107	90	197	-0.048309	0.056551	0.037315
12	106	87	193	-0.020305	0.057134	0.037560
13	104	84	188	-0.025907	0.057888	0.038090
14	102	83	185	-0.015957	0.058356	0.038450
15	98	79	177	-0.043243	0.059660	0.039555
16	96	76	172	-0.028249	0.060521	0.039973
17	94	75	169	-0.017442	0.061056	0.040282
18	86	74	160	-0.053254	0.062750	0.041630
19	85	72	157	-0.018750	0.063346	0.041812
20	85	73	158	+0.006369	0.063145	0.041949
21	85	76	161	+0.018987	0.062554	0.042262
22	83	75	158	-0.018634	0.063145	0.042645
23	80	69	149	-0.056962	0.065025	0.044097
24	77	65	142	-0.046980	0.066608	0.045289
25	74	61	135	-0.049296	0.068313	0.046268
Promedio	100.52	86.96	187.48	-0.009811	0.058571	0.039095

Tabla de la simulación con dos hijos, crossover uniforme, mutación en un punto, selección con el método de la rueda de la ruleta, sin elitismo, sin exigir mutación.

Tabla IV

El decrecimiento promedio es de 0.98%, lo que realmente es aceptable, si lo que se desea es disminuir la población o mantenerla, ya que el número inicial fue 179 individuos y el número al final de la simulación fue 135 individuos.



Tamaño poblacional de la segunda simulación con dos hijos

Figura 5.2

Al parecer no afecta en nada el tipo de crossover que se utiliza, aunque en realidad, esto hace que para cada individuo seleccionado, o par de individuos seleccionados en este caso, se busque la posibilidad de crear uno nuevo, con una aptitud mayor, debido a lo aleatorio del crossover.

Mutación

La mutación es un operador genético de baja probabilidad, por lo que espero que los resultados obtenidos sean similares a los de la primera simulación en la Tabla III, por lo que seguramente no se

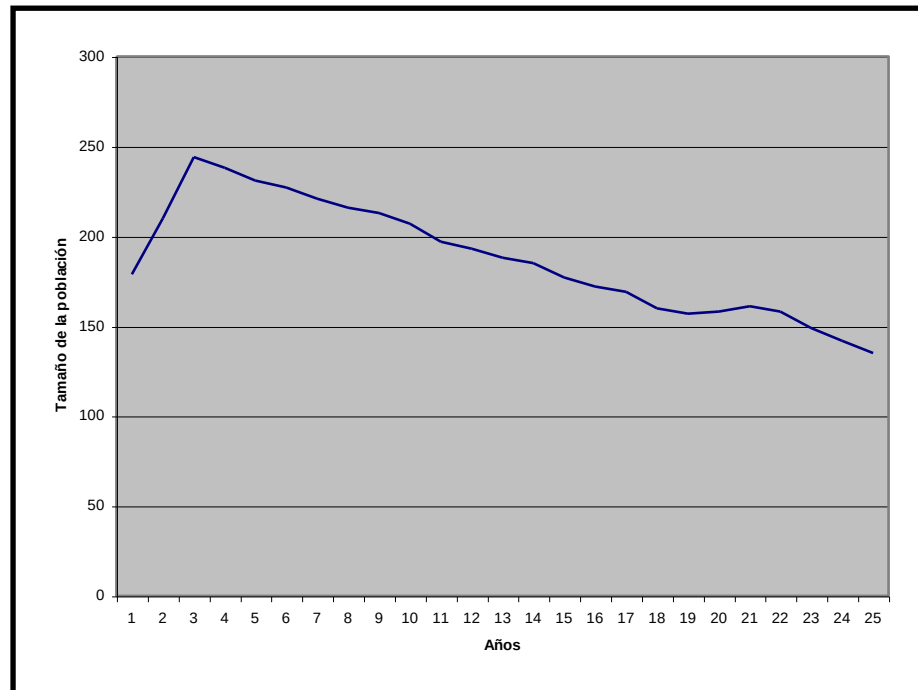
realiza, excepto en unas cuantas iteraciones. Cambiaré la mutación de un punto a la mutación uniforme, y observaremos los resultados en la siguiente tabla.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	94	90	184	+0.000000	0.058514	0.039127
2	108	105	213	+0.157609	0.054385	0.036104
3	120	122	242	+0.136150	0.051023	0.033324
4	119	121	240	-0.008264	0.051235	0.033632
5	118	119	237	-0.012500	0.051558	0.033803
6	117	117	234	-0.012658	0.051887	0.034088
7	115	116	231	-0.012821	0.052223	0.034221
8	113	113	226	-0.021645	0.052798	0.034673
9	110	108	218	-0.035398	0.053758	0.035484
10	107	107	214	-0.018349	0.054258	0.035684
11	105	105	210	-0.018692	0.054772	0.035978
12	104	103	207	-0.014286	0.055168	0.036037
13	102	96	198	-0.043478	0.056408	0.036980
14	98	94	192	-0.030303	0.057282	0.037340
15	96	92	188	-0.020833	0.057888	0.037600
16	92	87	179	-0.047872	0.059326	0.038443
17	83	86	169	-0.055866	0.061056	0.039217
18	86	89	175	+0.035503	0.060000	0.038329
19	85	90	175	+0.000000	0.060000	0.038229
20	85	91	176	+0.005714	0.059829	0.038159
21	86	90	176	+0.000000	0.059829	0.038574
22	83	87	170	-0.034091	0.060876	0.039095
23	81	83	164	-0.035294	0.061980	0.039709
24	79	82	161	-0.018293	0.062554	0.040373
25	74	81	155	-0.037267	0.063754	0.041223
Promedio	98.4	98.96	197.36	-0.005717	0.056894	0.037017

Tabla de la simulación con dos hijos, crossover en un punto, mutación uniforme, selección con el método de la rueda de la ruleta, sin elitismo, sin exigir mutación.

Tabla V

El decrecimiento fue en promedio del 0.571%, y el número de personas inicial fue 184 contra 155 que se obtuvo como número final. Son valores bastante buenos para nuestros propósitos de mantener la población.



Tamaño poblacional de la tercera simulación con dos hijos

Figura 5.3

En la ejecución utilizando mutación uniforme, se obtuvieron resultados similares a los de la tabla III, donde la mutación se escogía en un solo punto, como ya mencioné, esto se debe a la baja probabilidad de este operador.

Elitismo

Ahora mostraré una comparación entre utilizar y el no utilizar elitismo. El elitismo es un operador de selección, que mantiene el estado sin cambiar la aptitud de los más aptos. Para ello escogeré

las mismas opciones de crossover y mutación en un solo punto, con las mismas probabilidades.

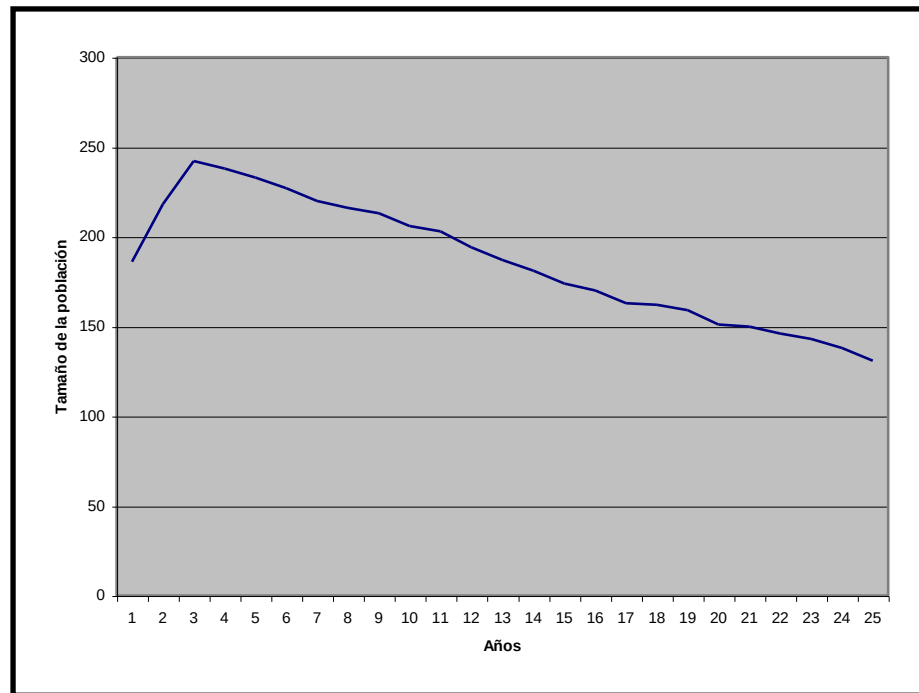
Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	94	92	186	+0.000000	0.058199	0.038292
2	111	107	218	+0.172043	0.053758	0.035197
3	124	118	242	+0.110092	0.051023	0.033592
4	122	116	238	-0.016529	0.051450	0.034059
5	120	113	233	-0.021008	0.051999	0.034289
6	117	110	227	-0.025751	0.052681	0.034622
7	113	107	220	-0.030837	0.053513	0.035236
8	110	106	216	-0.018182	0.054006	0.035457
9	109	104	213	-0.013889	0.054385	0.035607
10	104	102	206	-0.032864	0.055301	0.036658
11	103	100	203	-0.014563	0.055709	0.036862
12	102	92	194	-0.044335	0.056986	0.037813
13	100	87	187	-0.036082	0.058043	0.038372
14	97	84	181	-0.032086	0.058997	0.038899
15	93	81	174	-0.038674	0.060172	0.039653
16	91	79	170	-0.022989	0.060876	0.040009
17	87	76	163	-0.041176	0.062169	0.040732
18	87	75	162	-0.006135	0.062361	0.040807
19	86	73	159	-0.018519	0.062947	0.041449
20	82	69	151	-0.050314	0.064592	0.042681
21	82	68	150	-0.006623	0.064807	0.042705
22	81	65	146	-0.026667	0.065689	0.044333
23	79	64	143	-0.020548	0.066375	0.044821
24	76	62	138	-0.034965	0.067566	0.045627
25	74	57	131	-0.050725	0.069348	0.046883
Promedio	97.76	88.28	186.04	-0.012853	0.058918	0.038986

Tabla de la simulación con dos hijos, crossover en un punto, mutación en un punto, selección con el método de la rueda de la ruleta, con elitismo, sin exigir mutación.

Tabla VI

El decrecimiento promedio es del orden del 1.28%, que ya es un poco alto. Además el número inicial de 186 individuos a

disminuido a 131, y la población podría desaparecer en un momento determinado.



Tamaño poblacional de la cuarta simulación con dos hijos

Figura 5.4

Al utilizar el elitismo, la población se observa un decrecimiento mayor que al no utilizarlo, esto es por la forma de escoger los padres, o sea por no seleccionar a los individuos más aptos, para el crossover.

Exigir mutación

Debe recordarse que la exigencia de la mutación hace que se realice la mutación en cada uno de los individuos, en cada una de

las iteraciones, sin importar la probabilidad ingresada para la mutación.

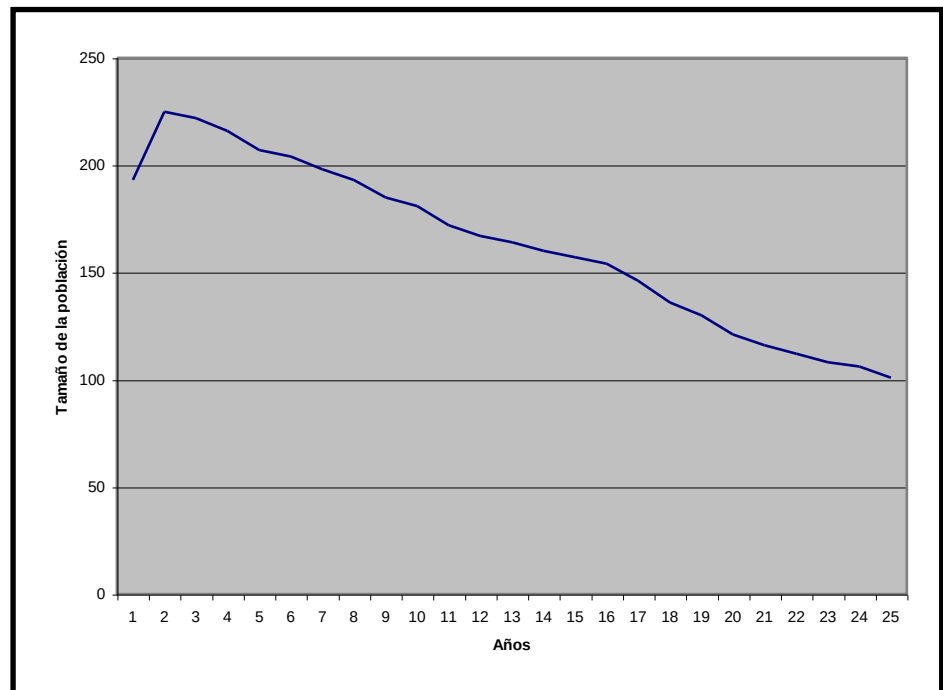
Utilizaré como referencia la Tabla III de la primera simulación, en la que no se exige la mutación, y luego veamos en la Tabla VII lo que sucede cuando exigimos la mutación para cada iteración.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	99	94	193	+0.000000	0.057134	0.039151
2	121	104	225	+0.165803	0.052915	0.035687
3	120	102	222	-0.013333	0.053271	0.035804
4	116	100	216	-0.027027	0.054006	0.036441
5	112	95	207	-0.041667	0.055168	0.037477
6	111	93	204	-0.014493	0.055572	0.037784
7	106	92	198	-0.029412	0.056408	0.038255
8	104	89	193	-0.025253	0.057134	0.038785
9	101	84	185	-0.041451	0.058356	0.039593
10	99	82	181	-0.021622	0.058997	0.040299
11	93	79	172	-0.049724	0.060521	0.041447
12	89	78	167	-0.029070	0.061420	0.041936
13	87	77	164	-0.017964	0.061980	0.042209
14	84	76	160	-0.024390	0.062750	0.042586
15	83	74	157	-0.018750	0.063346	0.042893
16	81	73	154	-0.019108	0.063960	0.043086
17	77	69	146	-0.051948	0.065689	0.044289
18	71	65	136	-0.068493	0.068061	0.045311
19	68	62	130	-0.044118	0.069614	0.046280
20	64	57	121	-0.069231	0.072157	0.047716
21	61	55	116	-0.041322	0.073696	0.048862
22	59	53	112	-0.034483	0.075000	0.049430
23	57	51	108	-0.035714	0.076376	0.050370
24	56	50	106	-0.018519	0.077093	0.050643
25	54	47	101	-0.047170	0.078979	0.051797
Promedio	86.92	76.04	162.96	-0.024738	0.063584	0.042725

Tabla de la simulación con dos hijos, crossover en un punto, mutación en un punto, selección con el método de la rueda de la ruleta, sin elitismo y exigiendo la mutación.

Tabla VII

Es notable como la población ha decrecido desde los 193 individuos iniciales hasta los 101 individuos de la última iteración. Debe recordarse que no se debe abusar del operador de la mutación y debe observarse un alto porcentaje de decrecimiento en la población, que en promedio es del 2.4%, lo que indica una baja tasa de natalidad o una alta tasa de mortalidad, y es realmente un decrecimiento acelerado.



Tamaño poblacional de la quinta simulación con dos hijos

Figura 5.5

Selección

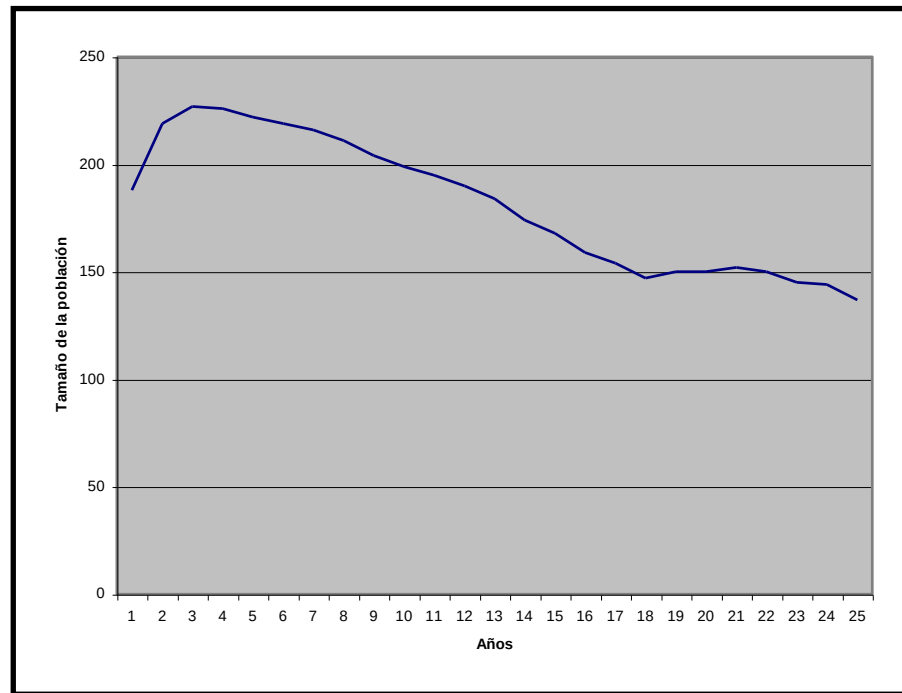
Ahora cambiaré del método de selección de la rueda de la ruleta que utilizamos en la Tabla # III que corresponde a la primera simulación y veamos en la Tabla VIII lo que sucede cuando aplicamos el modelo de selección del torneo.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	96	92	188	+0.000000	0.057888	0.039062
2	113	106	219	+0.164894	0.053635	0.036059
3	117	110	227	+0.036530	0.052681	0.035627
4	118	108	226	-0.004405	0.052798	0.035539
5	116	106	222	-0.017699	0.053271	0.035821
6	115	104	219	-0.013514	0.053635	0.035987
7	114	102	216	-0.013699	0.054006	0.036200
8	111	100	211	-0.023148	0.054642	0.036497
9	105	99	204	-0.033175	0.055572	0.037117
10	102	97	199	-0.024510	0.056266	0.037726
11	101	94	195	-0.020101	0.056840	0.038188
12	98	92	190	-0.025641	0.057583	0.038785
13	93	91	184	-0.031579	0.058514	0.039696
14	87	87	174	-0.054348	0.060172	0.040508
15	86	82	168	-0.034483	0.061237	0.041136
16	82	77	159	-0.053571	0.062947	0.042067
17	78	76	154	-0.031447	0.063960	0.042502
18	75	72	147	-0.045455	0.065465	0.043669
19	76	74	150	+0.020408	0.064807	0.043641
20	76	74	150	+0.000000	0.064807	0.043304
21	75	77	152	+0.013333	0.064380	0.042153
22	76	74	150	-0.013158	0.064807	0.042900
23	73	72	145	-0.033333	0.065915	0.043756
24	72	72	144	-0.006897	0.066144	0.044250
25	69	68	137	-0.048611	0.067813	0.045014
Promedio	92.96	88.24	181.20	-0.011744	0.059591	0.039888

Tabla de la simulación con dos hijos, crossover en un punto, mutación en un punto, selección con el método del torneo, sin elitismo y sin exigir la mutación.

Tabla VIII

Realmente en el método del torneo existe una mayor aleatoriedad, ya que no es proporcional a la aptitud de los individuos, sino que lo hace prácticamente de manera aleatoria uniforme.



Tamaño poblacional de la sexta simulación con dos hijos

Figura 5.6

Es notable que la población ha decrecido en promedio en 1.17%, y esto es por la forma de selección que utiliza el método del torneo, de tal manera, que el tamaño promedio de la población quedó atrás hace mucho, entre la iteración # 12 y la # 13. Además se puede observar que el comportamiento de la mejor aptitud y de la aptitud promedio es diferente, ya que la mejor aptitud está muy por

encima de la aptitud promedio. Otro detalle que debe observarse es la forma como cambia la curva en el gráfico, a diferencia de todos los anteriores.

Considerando cuatro hijos

Crossover

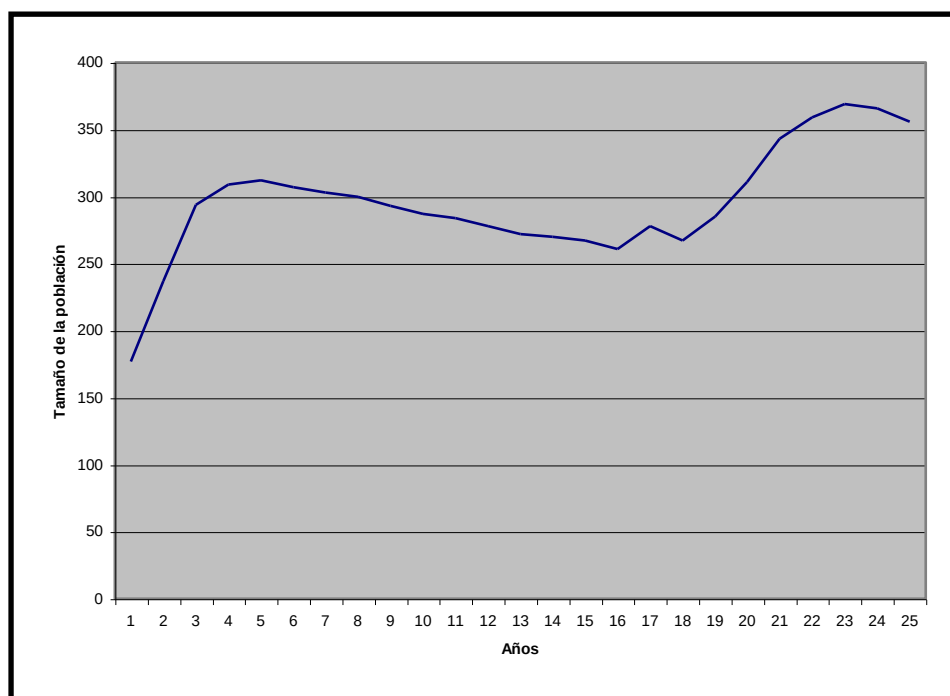
Realizaré dos ejecuciones comparando el crossover en un punto y el crossover uniforme.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	91	86	177	+0.000000	0.059660	0.040950
2	118	119	237	+0.338983	0.051558	0.034650
3	145	149	294	+0.240506	0.046291	0.031230
4	153	156	309	+0.051020	0.045153	0.030684
5	155	157	312	+0.009709	0.044936	0.030518
6	152	155	307	-0.016026	0.045300	0.030744
7	150	153	303	-0.013029	0.045598	0.031045
8	149	151	300	-0.009901	0.045826	0.031231
9	145	148	293	-0.023333	0.046370	0.031631
10	144	143	287	-0.020478	0.046852	0.031909
11	142	142	284	-0.010453	0.047099	0.032093
12	140	138	278	-0.021127	0.047604	0.032589
13	137	135	272	-0.021583	0.048127	0.032840
14	136	134	270	-0.007353	0.048305	0.032935
15	134	133	267	-0.011111	0.048575	0.033078
16	131	130	261	-0.022472	0.049130	0.033687
17	140	138	278	+0.065134	0.047604	0.032056
18	134	133	267	-0.039568	0.048575	0.032870
19	140	145	285	+0.067416	0.047016	0.031447
20	153	158	311	+0.091228	0.045008	0.030307
21	166	177	343	+0.102894	0.042857	0.028685
22	177	182	359	+0.046647	0.041891	0.027988
23	183	186	369	+0.027855	0.041320	0.027655
24	182	184	366	-0.008130	0.041489	0.027640
25	178	178	356	-0.027322	0.042067	0.028006
Promedio	147.00	148.40	295.40	+0.031580	0.046568	0.031539

Tabla de la simulación con cuatro hijos, crossover en un punto, mutación en un punto, selección con el método de la rueda de la ruleta, sin elitismo, sin exigir mutación.

Tabla IX

Debe notarse que la población continuará creciendo indefinidamente. En esta ejecución se tuvo un crecimiento promedio del 3,15% en 25 años.



Tamaño poblacional de la primera simulación con cuatro hijos

Figura 5.7

Notamos que a medida que la población crece, la aptitud promedio de la misma disminuye, es decir que aumentan los recursos para cada individuo de la población.

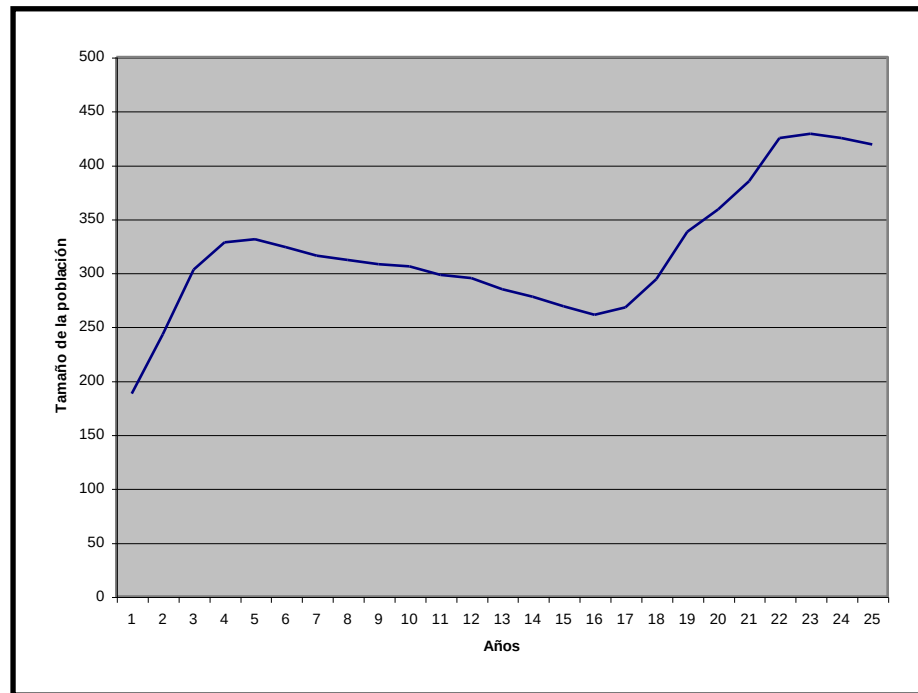
En la siguiente ejecución, utilicé crossover uniforme, y continué utilizando el modelo de selección de la rueda de la ruleta, y la mutación en un solo punto y obtuve los siguientes resultados.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	96	92	188	+0.000000	0.057888	0.039629
2	119	124	243	+0.292553	0.050918	0.034715
3	150	153	303	+0.246914	0.045598	0.031385
4	167	161	328	+0.082508	0.043826	0.030310
5	171	160	331	+0.009146	0.043627	0.030233
6	167	157	324	-0.021148	0.044096	0.030599
7	163	153	316	-0.024691	0.044651	0.030885
8	161	151	312	-0.012658	0.044936	0.031072
9	158	150	308	-0.012821	0.045227	0.031248
10	157	149	306	-0.006494	0.045374	0.031268
11	154	144	298	-0.026144	0.045979	0.031695
12	152	143	295	-0.010067	0.046212	0.031782
13	148	137	285	-0.033898	0.047016	0.032204
14	144	134	278	-0.024561	0.047604	0.032501
15	140	129	269	-0.032374	0.048394	0.033070
16	137	124	261	-0.029740	0.049130	0.033516
17	140	128	268	+0.026820	0.048484	0.032921
18	153	141	294	+0.097015	0.046291	0.031709
19	171	167	338	+0.149660	0.043173	0.029403
20	180	179	359	+0.062130	0.041891	0.028623
21	191	194	385	+0.072423	0.040452	0.027805
22	205	220	425	+0.103896	0.038501	0.026411
23	205	224	429	+0.009412	0.038321	0.026142
24	203	222	425	-0.009324	0.038501	0.026391
25	199	220	419	-0.014118	0.038776	0.026492
Promedio	161.24	158.24	319.48	+0.035778	0.044995	0.030880

Tabla de la simulación con cuatro hijos, crossover uniforme, mutación en un punto, selección con el método de la rueda de la ruleta, sin elitismo, sin exigir mutación.

Tabla X

El crecimiento poblacional utilizando el crossover de tipo uniforme es casi el mismo, es decir 3.57%, aunque el comportamiento de la curva en la gráfica fue un poco diferente al de utilizar crossover en un punto.



Tamaño poblacional de la segunda simulación con cuatro hijos

Figura 5.8

Realizar la misma comparación utilizando una mutación de tipo uniforme, no alteraría mayormente los resultados, ya que el proceso de la mutación recibe una muy pequeña probabilidad, que extrañamente se cumplirá.

Elitismo

Ahora mostraré una comparación entre el utilizar y el no utilizar elitismo. Para ello escogeré las mismas opciones de crossover y

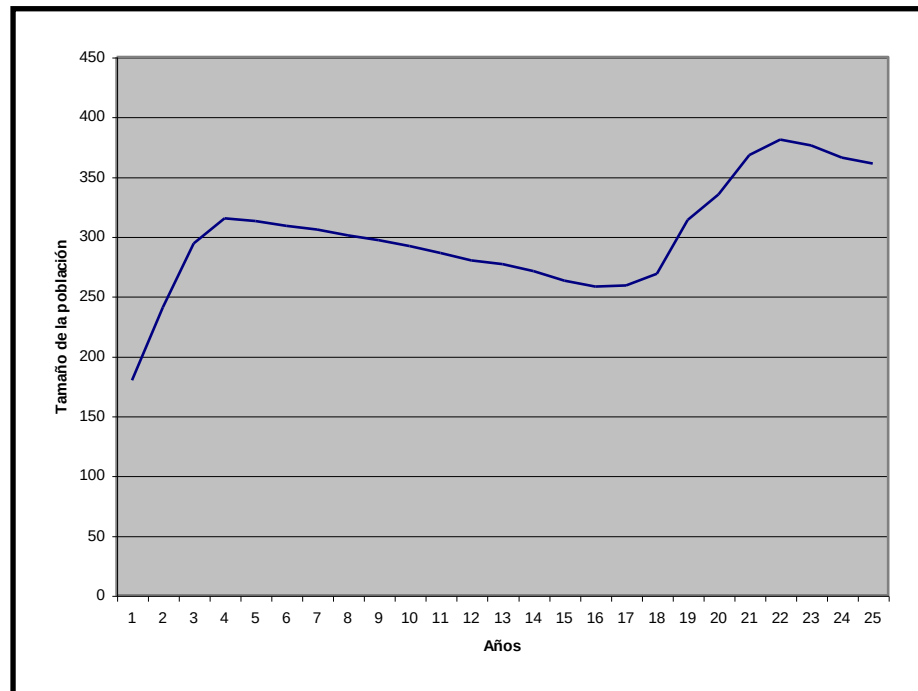
mutación en un solo punto, con las mismas probabilidades, y utilizaré la misma configuración utilizada hasta ahora para el modelo poblacional.

La Tabla sin elitismo es la Tabla IX de la primera simulación con cuatro hijos, y a continuación muestro en la Tabla XI la configuración con elitismo.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	89	91	180	+0.000000	0.059161	0.037984
2	116	125	241	+0.338889	0.051128	0.033524
3	147	147	294	+0.219917	0.046291	0.030435
4	155	160	315	+0.071429	0.044721	0.029227
5	154	159	313	-0.006349	0.044864	0.029223
6	152	157	309	-0.012780	0.045153	0.029317
7	150	156	306	-0.009709	0.045374	0.029514
8	149	152	301	-0.016340	0.045750	0.029830
9	146	151	297	-0.013289	0.046057	0.030105
10	143	149	292	-0.016835	0.046449	0.030314
11	139	147	286	-0.020548	0.046934	0.030478
12	135	145	280	-0.020979	0.047434	0.030574
13	133	144	277	-0.010714	0.047690	0.030765
14	130	141	271	-0.021661	0.048215	0.031047
15	126	137	263	-0.029520	0.048943	0.031470
16	123	135	258	-0.019011	0.049415	0.031666
17	119	140	259	+0.003876	0.049320	0.031598
18	121	148	269	+0.038610	0.048394	0.031109
19	146	168	314	+0.167286	0.044793	0.028916
20	154	181	335	+0.066879	0.043366	0.028178
21	171	197	368	+0.098507	0.041376	0.026699
22	178	203	381	+0.035326	0.040664	0.026168
23	175	201	376	-0.013123	0.040933	0.026405
24	170	196	366	-0.026596	0.041489	0.026959
25	169	192	361	-0.013661	0.041775	0.027092
Promedio	143.60	156.88	300.48	0.031584	0.046228	0.029944

Tabla de la simulación con cuatro hijos, crossover en un punto, mutación en un punto, selección con el método de la rueda de la ruleta, con elitismo, sin exigir mutación.

Tabla XI



Tamaño poblacional de la tercera simulación con cuatro hijos

Figura 5.9

Al utilizar el elitismo, la población se observa un crecimiento menor que al no utilizarlo. Aunque realmente lo que se observa es un crecimiento posterior a un decrecimiento. El crecimiento promedio es del orden del 3.15%, esto es por no seleccionar a los individuos más aptos, para el crossover y la mutación.

Exigir mutación

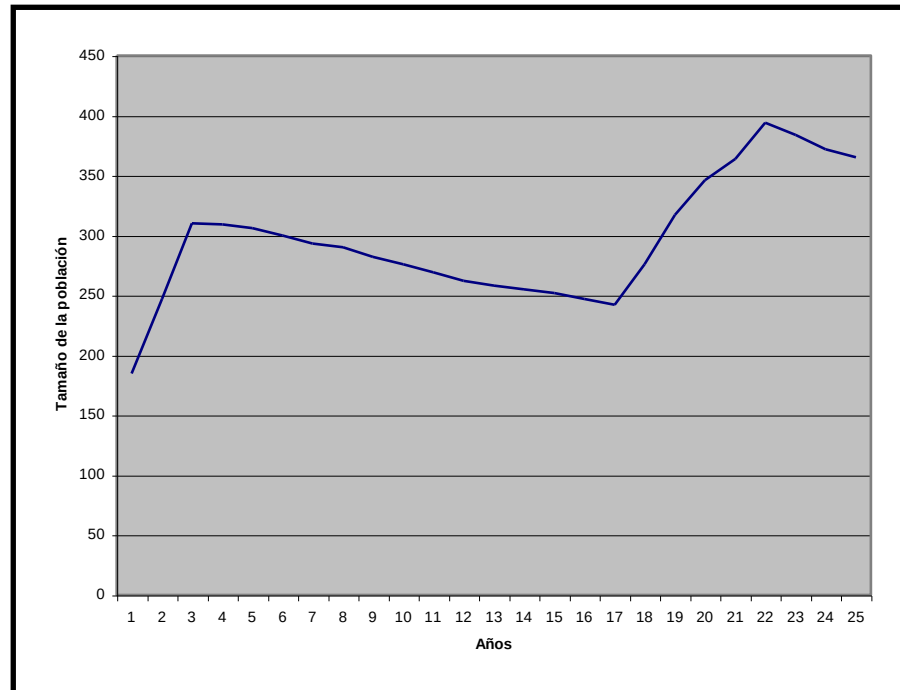
Utilizaré como referencia la tabla # 1, en la que no se exige la mutación, y luego veamos en la tabla # 10 lo que sucede cuando exigimos la mutación para cada iteración.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	96	89	185	+0.000000	0.058356	0.038071
2	127	120	247	+0.335135	0.050504	0.032895
3	156	154	310	+0.255061	0.045081	0.028729
4	153	156	309	-0.003226	0.045153	0.028782
5	152	154	306	-0.009709	0.045374	0.028901
6	149	151	300	-0.019608	0.045826	0.029258
7	147	146	293	-0.023333	0.046370	0.029413
8	146	144	290	-0.010239	0.046609	0.029569
9	140	142	282	-0.027586	0.047266	0.029902
10	137	139	276	-0.021277	0.047777	0.030221
11	134	135	269	-0.025362	0.048394	0.030624
12	131	131	262	-0.026022	0.049037	0.031005
13	128	130	258	-0.015267	0.049415	0.031133
14	126	129	255	-0.011628	0.049705	0.031270
15	124	128	252	-0.011765	0.050000	0.031432
16	123	124	247	-0.019841	0.050504	0.031810
17	119	123	242	-0.020243	0.051023	0.032190
18	136	140	276	+0.140496	0.047777	0.030382
19	157	160	317	+0.148551	0.044580	0.028426
20	167	179	346	+0.091483	0.042671	0.027172
21	175	189	364	+0.052023	0.041603	0.026323
22	187	207	394	+0.082418	0.039987	0.025138
23	180	204	384	-0.025381	0.040505	0.025388
24	174	198	372	-0.031250	0.041153	0.025774
25	171	194	365	-0.018817	0.041545	0.025847
Promedio	145.40	150.64	296.04	+0.031385	0.046649	0.029586

Tabla de la simulación con cuatro hijos, crossover en un punto, mutación en un punto, selección con el método de la rueda de la ruleta, sin elitismo, con exigencia de la mutación.

Tabla XII

Debe observarse un bajo porcentaje de crecimiento en la población, que en promedio es del 3.13%, lo que indica una baja tasa de natalidad o una alta tasa de mortalidad. Debe observarse también la curva en la gráfica, que se ve un poco rígida y escalonada.



Tamaño poblacional de la cuarta simulación con cuatro hijos

Figura 5.10

Selección

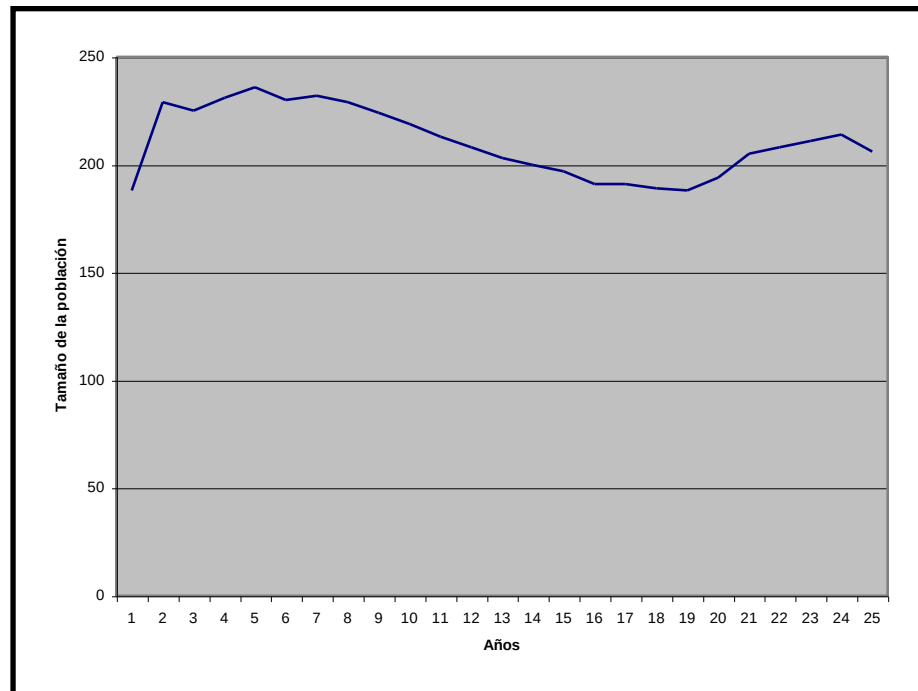
Ahora cambiaré del método de selección de la rueda de la ruleta que utilizamos en la Tabla IX de la primera simulación con cuatro hijos y veamos en la Tabla XIII lo que sucede cuando aplicamos el modelo de selección del torneo. En este método de selección los individuos están ordenados por aptitud de mayor a menor; pero reciben una probabilidad de ser escogidos que es casi igual para todos, sin importar la aptitud.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	97	91	188	+0.000000	0.057888	0.036235
2	122	107	229	+0.218085	0.052451	0.033729
3	120	105	225	-0.017467	0.052915	0.034062
4	119	112	231	+0.026667	0.052223	0.034419
5	121	115	236	+0.021645	0.051667	0.033893
6	117	113	230	-0.025424	0.052337	0.034201
7	117	115	232	+0.008696	0.052111	0.033878
8	115	114	229	-0.012931	0.052451	0.034464
9	111	113	224	-0.021834	0.053033	0.034873
10	108	111	219	-0.022321	0.053635	0.035103
11	105	108	213	-0.027397	0.054385	0.035651
12	104	104	208	-0.023474	0.055035	0.036373
13	102	101	203	-0.024038	0.055709	0.037112
14	100	100	200	-0.014778	0.056125	0.037262
15	96	101	197	-0.015000	0.056551	0.037738
16	95	96	191	-0.030457	0.057432	0.038386
17	98	93	191	+0.000000	0.057432	0.038705
18	96	93	189	-0.010471	0.057735	0.038294
19	97	91	188	-0.005291	0.057888	0.038249
20	100	94	194	+0.031915	0.056986	0.038033
21	102	103	205	+0.056701	0.055436	0.036997
22	101	107	208	+0.014634	0.055035	0.036941
23	102	109	211	+0.014423	0.054642	0.037043
24	103	111	214	+0.014218	0.054258	0.036838
25	100	106	206	-0.037383	0.055301	0.037454
Promedio	105.92	104.52	210.44	+0.004749	0.054826	0.036237

Tabla de la simulación con cuatro hijos, crossover en un punto, mutación en un punto, selección con el método del torneo, sin elitismo, sin exigir de la mutación.

Tabla XIII

La población ha crecido en promedio 4.75%, y esto es por la forma de selección que utiliza el método del torneo y debe observarse la forma como cambia la curva en el gráfico, a diferencia de todos los anteriores.



Tamaño poblacional de la quinta simulación con cuatro hijos

Figura 5.11

Puede observarse que en todas las configuraciones, el número de individuos crece y decrece periódicamente, o crece una vez y luego decrece o viceversa; pero nunca tiene un comportamiento monótono pese a que su tendencia sí lo es.

En realidad estas son las variaciones en la configuración del algoritmo genético; pero también puede hacerse variar los parámetros de la población, así como el límite de vida, o el intervalo de fecundidad de los individuos. Entonces a continuación

realizaré dos simulaciones más para observar el comportamiento de la curva en estas condiciones.

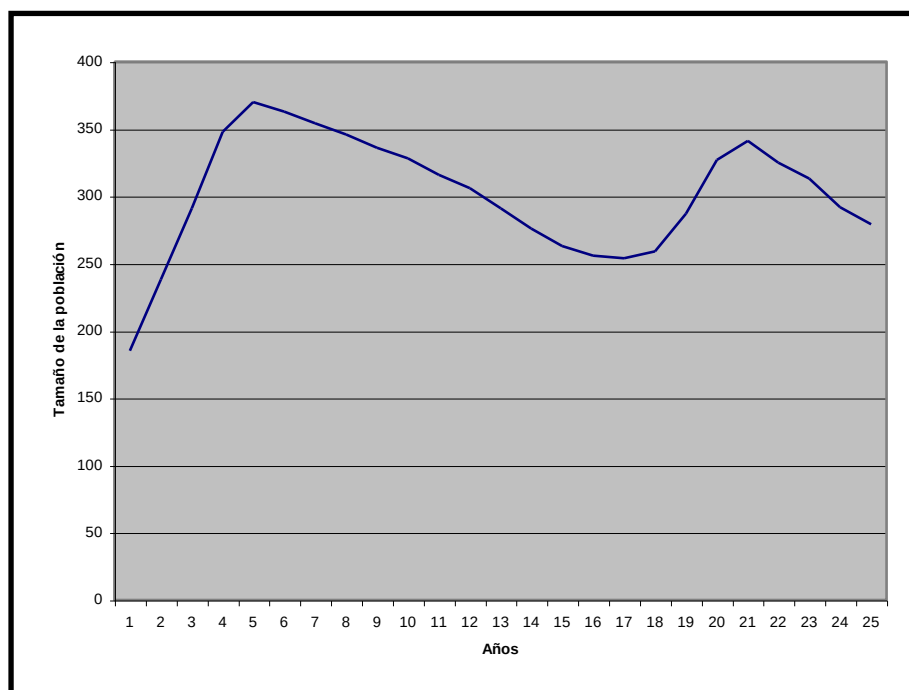
Edad límite de vida

A continuación utilizaré la misma configuración de la tabla uno, con cuatro hijos; pero disminuiré la edad límite de vida a $w=50$ para observar que pasa con el crecimiento poblacional.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	91	94	185	+0.000000	0.058356	0.040145
2	113	125	238	+0.286486	0.051450	0.034874
3	143	148	291	+0.222689	0.046529	0.031219
4	170	178	348	+0.195876	0.042548	0.028648
5	180	190	370	+0.063218	0.041264	0.027597
6	176	187	363	-0.018919	0.041660	0.027933
7	170	184	354	-0.024793	0.042186	0.028314
8	166	180	346	-0.022599	0.042671	0.028577
9	162	174	336	-0.028902	0.043301	0.029001
10	159	169	328	-0.023810	0.043826	0.029441
11	156	160	316	-0.036585	0.044651	0.029909
12	150	156	306	-0.031646	0.045374	0.030266
13	143	148	291	-0.049020	0.046529	0.030920
14	136	140	276	-0.051546	0.047777	0.031510
15	132	131	263	-0.047101	0.048943	0.032560
16	127	129	256	-0.026616	0.049608	0.032934
17	131	123	254	-0.007813	0.049803	0.032532
18	134	125	259	+0.019685	0.049320	0.032003
19	152	135	287	+0.108108	0.046852	0.030486
20	175	152	327	+0.139373	0.043893	0.028689
21	184	157	341	+0.042813	0.042983	0.028032
22	177	148	325	-0.046921	0.044028	0.028474
23	173	140	313	-0.036923	0.044864	0.028798
24	163	129	292	-0.067093	0.046449	0.029728
25	156	123	279	-0.044521	0.047519	0.030387
Promedio	152.76	149.00	301.76	+0.020538	0.046095	0.030519

Tabla de la simulación con el límite de vida disminuido a 50 años.

Tabla XIV



Tamaño poblacional de la sexta simulación con cuatro hijos

Figura 5.12

La tabla muestra un crecimiento muy pequeño, del orden del 2.05%, y el gráfico indica la forma acelerada como crece y luego decrece y crece la población debido a la poca edad que los individuos pueden tener, y no se reproducen en forma completa.

Intervalo de edades de reproducción

A continuación utilizaré la misma configuración de la Tabla IX, con cuatro hijos; pero disminuiré las edades de reproducción para observar que sucede con el crecimiento poblacional. Utilizaré las edades desde los 25 hasta los 30 años únicamente.

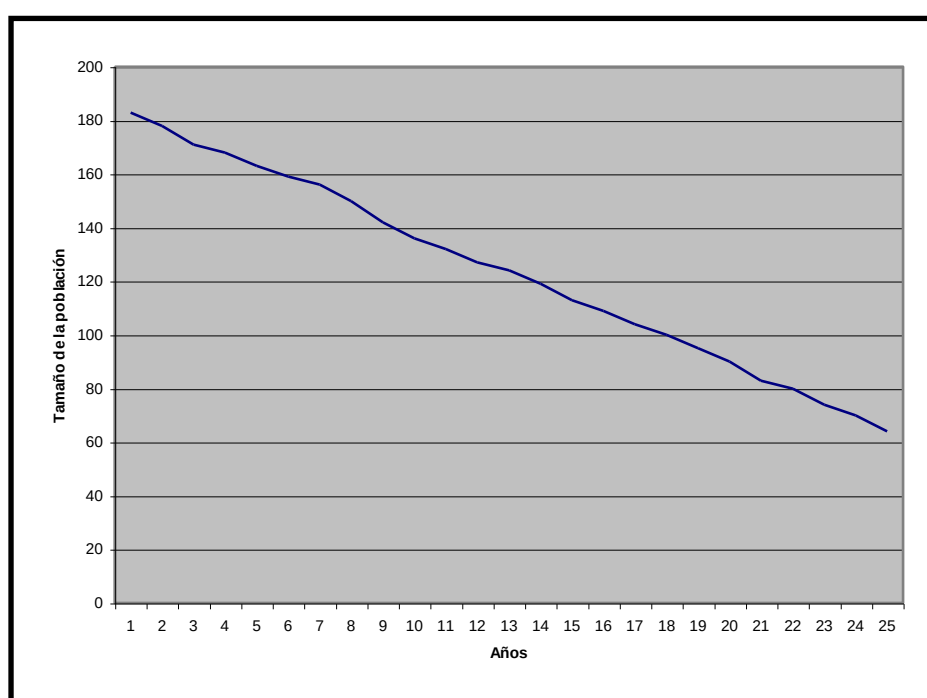
Evidentemente, esto no permitirá si se utiliza en promedio una cantidad mayor a cinco hijos, que todos lleguen a nacer y a pertenecer a la población, ya que se trunca el crecimiento en la edad límite superior.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	89	94	183	+0.000000	0.058674	0.039829
2	87	91	178	-0.027322	0.059492	0.040349
3	83	88	171	-0.039326	0.060698	0.040881
4	81	87	168	-0.017544	0.061237	0.041401
5	78	85	163	-0.029762	0.062169	0.042155
6	77	82	159	-0.024540	0.062947	0.042865
7	76	80	156	-0.018868	0.063549	0.043508
8	73	77	150	-0.038462	0.064807	0.045004
9	68	74	142	-0.053333	0.066608	0.045886
10	65	71	136	-0.042254	0.068061	0.047043
11	63	69	132	-0.029412	0.069085	0.047757
12	62	65	127	-0.037879	0.070432	0.048800
13	61	63	124	-0.023622	0.071279	0.050177
14	59	60	119	-0.040323	0.072761	0.051161
15	56	57	113	-0.050420	0.074667	0.052789
16	53	56	109	-0.035398	0.076025	0.053480
17	51	53	104	-0.045872	0.077831	0.054482
18	48	52	100	-0.038462	0.079373	0.055179
19	46	49	95	-0.050000	0.081435	0.057122
20	44	46	90	-0.052632	0.083666	0.057721
21	39	44	83	-0.077778	0.087123	0.060518
22	38	42	80	-0.036145	0.088034	0.061039
23	35	39	74	-0.075000	0.091533	0.063092
24	32	38	70	-0.054054	0.094112	0.065510
25	30	34	64	-0.085714	0.098425	0.068938
Promedio	59.76	63.84	123.6	-0.040965	0.073761	0.051067

Tabla de la simulación con el intervalo de fecundidad disminuido desde los 25 a los 30 años únicamente.

Tabla XV

La población ha decrecido en promedio en 4.09% en 25 años, lo cual indica que por más que haya utilizado cuatro hijos, estos no alcanzan a nacer todos. Esto indica que los hijos no necesariamente nacen en años consecutivos en este modelo poblacional, que fue diseñado mediante algoritmos genéticos.



Tamaño poblacional de la sexta simulación con cuatro hijos

Figura 5.13

La gráfica presenta un decrecimiento sumamente acelerado y casi de forma lineal, y aunque con cuatro hijos existe un supuesto de crecimiento de la población, en este caso no se cumple.

Límite de la población

Otra opción que resulta interesante es limitar el crecimiento de la población, agregando un valor límite, que significa la capacidad máxima de la población.

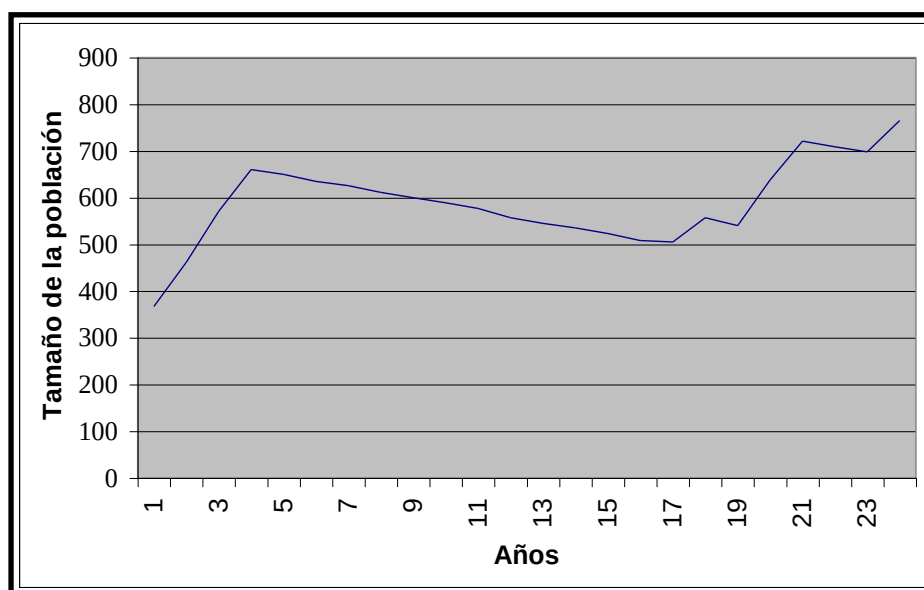
En mi aplicación un valor de cero utilizado hasta ahora implica que no hay límites; pero si lo utilizamos, se estabilizará.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.
1	184	183	367	+0.000000	0.041432	0.027699
2	231	231	462	+0.258856	0.036927	0.024486
3	297	274	571	+0.235931	0.033216	0.021841
4	339	321	660	+0.155867	0.030896	0.020273
5	335	315	650	-0.015152	0.031132	0.020402
6	329	306	635	-0.023077	0.031498	0.020698
7	323	303	626	-0.014173	0.031724	0.020864
8	316	295	611	-0.023962	0.032111	0.021097
9	310	290	600	-0.018003	0.032404	0.021249
10	305	284	589	-0.018333	0.032705	0.021481
11	298	279	577	-0.020374	0.033043	0.021718
12	289	268	557	-0.034662	0.033631	0.022067
13	283	262	545	-0.021544	0.033999	0.022256
14	276	259	535	-0.018349	0.034316	0.022546
15	271	252	523	-0.022430	0.034707	0.022741
16	262	246	508	-0.028681	0.035216	0.023099
17	263	242	505	-0.005906	0.035320	0.023027
18	283	274	557	+0.102970	0.033631	0.021802
19	276	264	540	-0.030521	0.034157	0.022147
20	333	305	638	+0.181481	0.031424	0.020267
21	380	341	721	+0.130094	0.029560	0.019167
22	373	336	709	-0.016644	0.029809	0.019352
23	368	330	698	-0.015515	0.030043	0.019543
24	405	360	765	+0.095989	0.028697	0.018607
25	396	354	750	-0.019608	0.028983	0.018827
Promedio	309.00	286.96	595.96	+0.032570	0.032823	0.021490

Tabla de la simulación limitando el crecimiento de la población

Tabla XVI

En esta ejecución puede notarse que las cantidades iniciales fueron cambiadas, esto es por efectos de observar mejor el crecimiento de la población, se escogieron valores un poco más altos, es decir 200 hombres y 200 mujeres. Además se agregó un valor límite de la población que son 700 personas.



Tamaño poblacional de la séptima simulación con cuatro hijos

Figura 5.14

El gráfico muestra crecimientos y decrecimientos a través del tiempo; pero este modelo está limitado a un máximo de 700 individuos, por lo que puede crecer un poco más de eso y luego decrecerá otra vez.

5.3. Predicción de los resultados por medio de los métodos convencionales.

En esta parte del proyecto, voy a utilizar una de las posibles configuraciones del Algoritmos genéticos – modelo poblacional, para compararla con los métodos convencionales de crecimientos poblacionales.

Utilizaré la configuración dada en la primera ejecución con cuatro hijos que se muestra en la tabla IX, esto es con probabilidades de mutación de 0.01 y crossover de 0.9, así como los métodos de un solo punto, y el modelo de selección de la ruleta, sin elitismo, y sin exigir la mutación; pero lo haré utilizando 48 iteraciones, para tomar un subconjunto de estas, de 45 iteraciones y tratar de comparar; para averiguar a que modelo se ajusta.

Lo haré además con la misma configuración del intervalo de edades de reproducción, es decir desde los 15 hasta los 44 años, y una edad límite de $w=80$ años. He considerado esta configuración, porque fue con la que comparé todas las demás, y porque los resultados son muy coherentes a la realidad.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.	Edades
01	4552	4563	9115	+0.000000	0.008314	0.005557	38.51
02	4733	4758	9491	+0.041251	0.008147	0.005435	36.61
03	4928	4971	9899	+0.042988	0.007978	0.005314	34.76
04	5177	5206	10383	+0.048894	0.007789	0.005187	32.86
05	5448	5458	10906	+0.050371	0.007600	0.005055	31.00
06	5739	5723	11462	+0.050981	0.007414	0.004916	29.42
07	5991	5960	11951	+0.042663	0.007261	0.004819	28.18
08	6220	6231	12451	+0.041838	0.007113	0.004717	26.99
09	6514	6482	12996	+0.043772	0.006963	0.004610	25.84
10	6831	6751	13582	+0.045091	0.006811	0.004504	24.79
11	6885	6805	13690	+0.007952	0.006784	0.004485	24.68
12	6728	6692	13420	-0.019722	0.006852	0.004529	25.29
13	6615	6568	13183	-0.017660	0.006913	0.004569	25.85
14	6470	6440	12910	-0.020708	0.006986	0.004617	26.30
15	6326	6305	12631	-0.021611	0.007062	0.004668	26.82
16	6174	6165	12339	-0.023118	0.007145	0.004722	27.28
17	6220	6195	12415	+0.006159	0.007124	0.004701	27.05

Tabla de las iteraciones 1 a 17 de la simulación para comparar

Tabla XVII

En esta tabla se presentan las primeras 17 iteraciones del algoritmo, correspondientes a los primeros 17 años de la población, nótese que la población creció hasta la iteración # 11 y luego decreció.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.	Edades
18	6248	6235	12483	+0.005477	0.007104	0.004687	26.80
19	6448	6483	12931	+0.035889	0.006980	0.004596	25.77
20	6716	6697	13413	+0.037275	0.006853	0.004515	24.84
21	7014	6952	13966	+0.041229	0.006716	0.004420	23.80
22	7269	7256	14525	+0.040026	0.006586	0.004329	22.97
23	7583	7524	15107	+0.040069	0.006458	0.004242	22.19
24	7848	7779	15627	+0.034421	0.006349	0.004161	21.57
25	8113	8024	16137	+0.032636	0.006248	0.004092	20.97
26	8387	8313	16700	+0.034889	0.006142	0.004018	20.46
27	8694	8646	17340	+0.038323	0.006028	0.003942	19.83
28	9048	8943	17991	+0.037543	0.005918	0.003874	19.28

Tabla de las iteraciones 18 a 28 de la simulación para comparar

Tabla XVIII

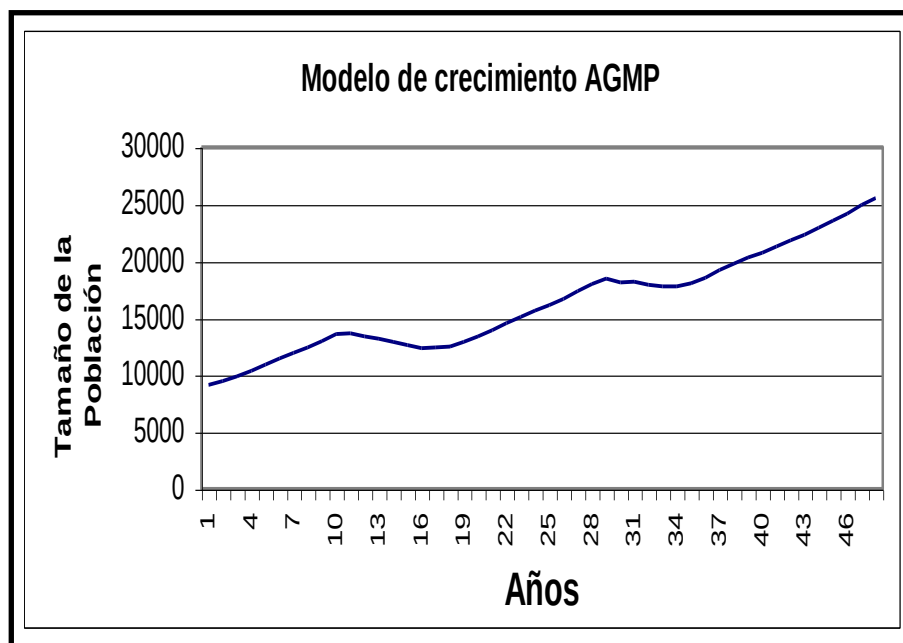
Se muestran las iteraciones desde la # 18 hasta la # 28, y se puede apreciar un crecimiento y luego un decrecimiento.

Año	Hombres	Mujeres	Total	Crecimiento	M. Aptitud	Aptitud P.	Edades
29	9310	9178	18488	+0.027625	0.005837	0.003814	18.92
30	9139	9011	18150	-0.018282	0.005892	0.003847	19.46
31	9143	9030	18173	+0.001267	0.005888	0.003844	19.65
32	9019	8900	17919	-0.013977	0.005929	0.003868	19.98
33	8919	8834	17753	-0.009264	0.005957	0.003883	20.37
34	8930	8845	17775	+0.001239	0.005953	0.003877	20.44
35	9046	8975	18021	+0.013840	0.005913	0.003853	20.34
36	9308	9250	18558	+0.029799	0.005826	0.003794	19.81
37	9668	9556	19224	+0.035887	0.005725	0.003725	19.40
38	9954	9810	19764	+0.028090	0.005646	0.003676	19.14
39	10226	10092	20318	+0.028031	0.005568	0.003623	18.86
40	10452	10313	20765	+0.022000	0.005508	0.003583	18.71
41	10707	10577	21284	+0.024994	0.005441	0.003531	18.52
42	10976	10852	21828	+0.025559	0.005372	0.003485	18.32
43	11209	11132	22341	+0.023502	0.005310	0.003443	18.15
44	11492	11407	22899	+0.024977	0.005245	0.003403	18.03
45	11849	11707	23556	+0.028691	0.005172	0.003356	17.91
46	12156	12034	24190	+0.026915	0.005103	0.003309	17.84
47	12540	12356	24896	+0.029186	0.005030	0.003260	17.77
48	12860	12705	25565	+0.026872	0.004964	0.003220	17.64

Tabla de las iteraciones 29 a 48 de la simulación para comparar

Tabla XIX

En esta tabla se observan las últimas iteraciones del algoritmo, es decir desde la iteración # 29 hasta la iteración #48, y se puede observar un crecimiento bastante acelerado, y es también evidente la forma como ha decrecido la función de aptitud.



Línea a través del tiempo de los datos simulados

Figura 5.15

Esta gráfica muestra el crecimiento y decrecimiento de la población, y por la misma, podemos observar que es convexa, por lo cual puede modelarse su tendencia mediante un modelo de crecimiento exponencial.

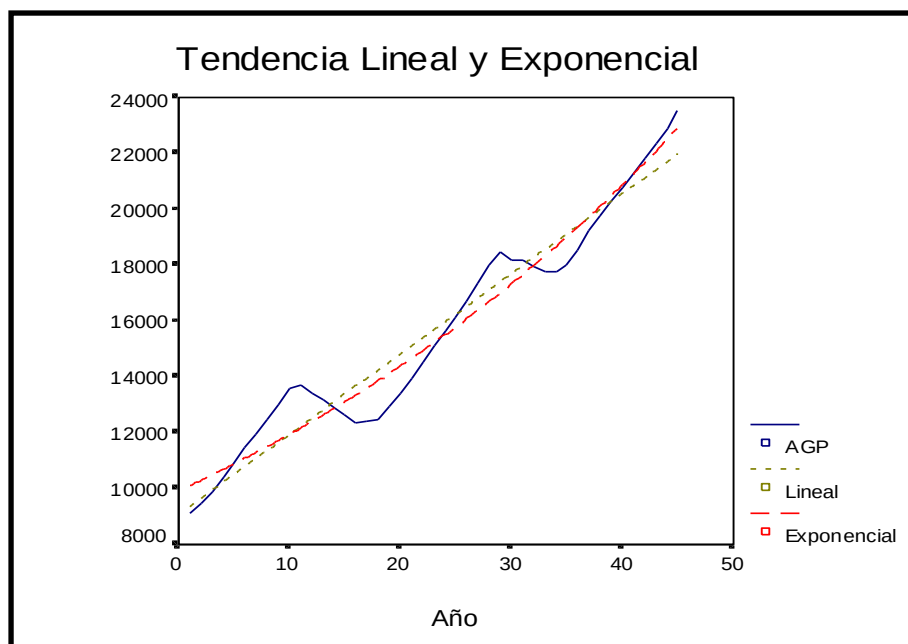
Utilizando el AGMP, en su parte del modelo exponencial mediante mínimos cuadrados obtuve la tendencia exponencial del modelo para las primeras 45 iteraciones, mediante la siguiente ecuación:

$$P(t) = 9918.01 e^{0.0186t}$$

De igual manera quise comparar con un crecimiento lineal, y obtuve la siguiente ecuación:

$$P(t) = 9084.09 + 287.030t$$

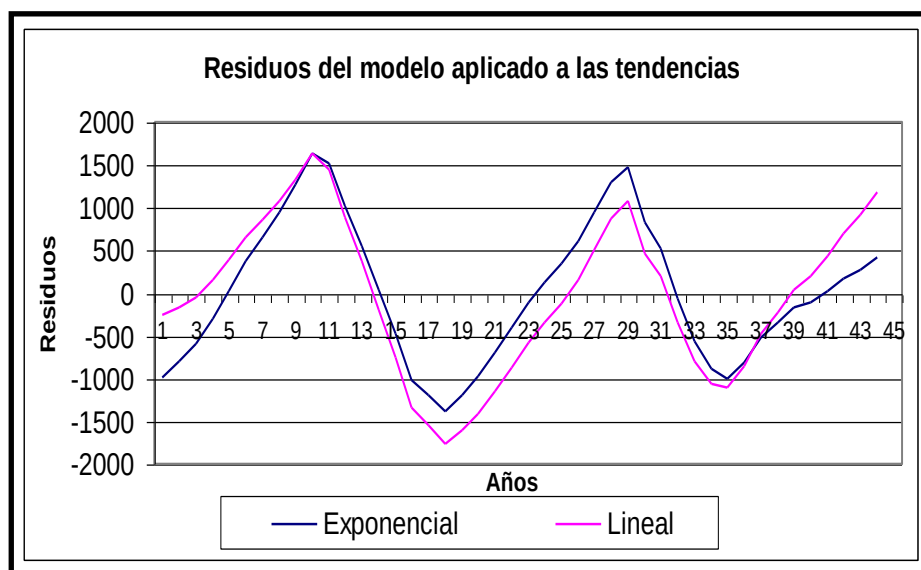
El modelo exponencial presenta un coeficiente de determinación $R^2 = 0.944$, que es el mismo que presenta el modelo lineal. Lo diferente es la media de cuadrados del error que presenta el exponencial con un valor de 0.0036034, mientras que el modelo lineal presenta una media de cuadrados del error de 861311.1, realmente grande. Además podemos observar en la siguiente gráfica como la tendencia exponencial explica el modelo un poco mejor que la tendencia lineal.



Comparación con las tendencias lineal y exponencial

Figura 5.16

Lo que sigue es analizar los errores en cada punto de la curva. Para ello veamos la gráfica de los errores del modelo exponencial y del modelo lineal.



Residuos de los datos ajustados a los modelos lineal y exponencial

Figura 5.17

Al realizar esta operación, de restarles la tendencia a los datos, en el modelo exponencial se obtiene una serie con oscilaciones acentuadas y con repeticiones cada 17.5 años. Con esto podría tratarse de predecir los tres valores que le quité a la serie antes de hallar su tendencia, y compararé para ver que tan precisa es la predicción. En realidad se observará lo malo de la predicción del modelo lineal. Los valores estimados por los modelos exponencial y lineal en los años 46, 47 y 48 se presentan en las siguientes tablas:

Año	AGMP	Límite Inferior	Exponencial	Límite Superior
46	24190	20567.68888	23341.23013	26488.78185
47	24896	20946.44648	23779.57903	26995.90975
48	25565	21331.87674	24226.16014	27513.13643

Valores estimados por el modelo exponencial

Tabla XX

Año	AGMP	Límite Inferior	Lineal	Límite Superior
46	24190	20331.70868	22287.46566	24243.22263
47	24896	20613.20080	22574.49556	24535.79033
48	25565	20894.47387	22861.52547	24828.57707

Valores estimados por el modelo lineal

Tabla XXI

Es evidente como un modelo lineal predice valores realmente malos, mientras que el modelo exponencial se acerca bastante a los valores reales. Lo importante es que están dentro del intervalo de 95% de confianza. Además debe recordarse que estamos analizando personas, por lo que se debe tratar con datos enteros, quedando la tabla para el modelo exponencial de la siguiente manera en los años 46, 4 y 48.

Año	AGMP	Límite Inferior	Exponencial	Límite Superior
46	24190	20567	23341	26488
47	24896	20946	23779	26995
48	25565	21331	24226	27513

Valores enteros estimados por el modelo lineal

Tabla XXII

5.4. Predicción de los resultados mediante algoritmos genéticos.

Primera ejecución

Al hacer uso de mi aplicación del modelo exponencial, obtuve los siguientes valores para β_0 y β_1 , haciendo una ejecución con un máximo de 50 iteraciones y 256 nodos.

Parámetro	Mínimos Cuadrados	AGMP
β_0	9918.01	9905.15
β_1	0.01863	0.01514

Primera estimación de B_0 y B_1 para el modelo exponencial

Tabla XXIII

Obteniéndose un modelo de la siguiente manera:

$$P(t) = 9905.15 e^{0.01514t}$$

Los valores obtenidos mediante los algoritmos genéticos no son los mismos que mostró el programa AGMP con mínimos cuadrados; pero son bastante cercanos, lo cual demuestra que los algoritmos genéticos tienen una convergencia que se aproxima bastante a los verdaderos valores.

Segunda ejecución

Para la segunda ejecución se realizó un máximo de 100 iteraciones y 1024 nodos, y se obtuvieron los siguientes resultados que se presentan en la siguiente tabla.

Parámetro	Mínimos Cuadrados	AGMP
β_0	9918.01	9915.43
β_1	0.01863	0.01790

Segunda estimación de B_0 y B_1 para el modelo exponencial

Tabla XXIV

Quedando el modelo de la siguiente forma:

$$P(t) = 9918.15 e^{0.0179t}$$

Los valores para esta ejecución fueron más cercanos a los que se obtuvieron con los mínimos cuadrados, esto es porque se aumentó el tamaño de la búsqueda.

Tercera ejecución

Finalmente, realicé una tercera ejecución, con un máximo de 200 iteraciones y 4096 nodos, y se obtuvieron los siguientes resultados.

Parámetro	Mínimos Cuadrados	AGMP
β_0	9918.01	9918.05
β_1	0.01863	0.01800

Tercera estimación de B_0 y B_1 para el modelo exponencial

Tabla XXV

El modelo finalmente queda casi igual al que mostró el método de los mínimos cuadrados:

$$P(t) = 9918.05 e^{0.0180t}$$

Los valores obtenidos para β_0 y β_1 para esta ejecución fueron casi los mismos que los que se obtuvo con los mínimos cuadrados. Lamentablemente el tiempo de ejecución aumentó en una forma muy considerable.

5.5. Selección del mejor modelo mediante algoritmos genéticos.

Para esta parte voy a utilizar un archivo de texto que contiene datos acerca de la natalidad, mortalidad, emigraciones y las inmigraciones, así como la variable explicada que es el tamaño de la población.

En la tabla XXVI se muestran los datos utilizados para realizar la regresión de tipo exponencial.

Tamaño	Nacimientos	Muertes	Emigraciones	Inmigraciones
920	13	18	17	3
927	8	2	7	18
929	10	17	13	1
940	9	6	8	4
957	1	5	15	10
974	5	5	4	19
993	17	16	13	8
993	11	14	9	19
1001	15	10	19	13
1018	6	6	14	17
1020	2	3	6	16
1024	18	10	11	19
1024	13	14	3	16
1024	8	3	18	1
1027	13	5	2	12
1031	1	1	17	6
1031	18	13	2	9
1036	0	10	14	11
1042	6	5	10	5
1053	19	18	18	17

Datos para realizar la ejecución de la selección del mejor modelo de varias variables.

Tabla XXVI

El usuario escogió el modelo en que sólo se utilizan las variables de natalidad y mortalidad, dejándose a un lado los valores de las emigraciones y de las inmigraciones; obteniéndose los siguientes resultados, junto con los que escogió el algoritmo genético, que tiene mejor explicación y menor suma de cuadrados del error. En la siguiente tabla se muestran los resultados de ambos modelos.

<i>Variable Aleatoria</i>	<i>AMGP</i>	<i>Elegido</i>
β_0	975.583455	998.427819
β_1	0.000381	0.001899
β_2	0	-0.002146
β_3	0	0
β_4	0.001641	0
SCE	0.03287996	0.03330770
R^2	0.51644115	0.51321272
Estadístico	9.07800550	8.96142607

Resultados de la ejecución del algoritmo genético para seleccionar el modelo

Tabla XXVI

5.6. Observaciones.

En este capítulo, he revisado los resultados de realizar una simulación poblacional utilizando los algoritmos genéticos, y luego ajusté los datos a un modelo exponencial, debido a la tendencia de los mismos, tratando inclusive de predecir valores que ya eran conocidos, y finalmente he comparado los resultados de la

ejecución de mi aplicación del modelo exponencial con algoritmos genéticos y con mínimos cuadrados.

Este simulador de crecimiento poblacional con algoritmos genéticos funciona con valores de aptitudes aleatorias tomadas del algoritmo genético, y así mismo con cada uno de sus parámetros, por lo que los resultados podrían variar de una corrida a otra. En las aplicaciones de los modelos exponencial y logístico, a mayor cantidad de nodos de búsqueda, darán una mejor aproximación de los resultados.

En el algoritmo genético de optimización para selección de un mejor modelo, el algoritmo dará un buen resultado, que probablemente no sea el mejor modelo que tengamos, pero sí es uno de los que más explican dichos resultados.

Considero importante aclarar que en estas simulaciones existen oscilaciones que en los modelos poblacionales no existen; pero esto se debe al tamaño inicial de la población. A medida que transcurre el tiempo, y la simulación avanza, las oscilaciones serán más pequeñas hasta desaparecer.

CONCLUSIONES

1. Los algoritmos genéticos permiten realizar una simulación muy versátil del crecimiento de una población, haciendo unas pequeñas modificaciones a los operadores del algoritmo genético básico y aumentando algunos operadores.
2. Al utilizar únicamente dos hijos en las simulaciones, la población se mantiene constante o decrece, mientras que al realizar las simulaciones con un mayor número de hijos, la población muestra un crecimiento.
3. En la simulación, el uso del crossover uniforme dará mejores resultados que el crossover en un sólo punto, ya que habrá una gran parte de la población que se reproducirá, mientras que en el crossover en un punto, al fallar la probabilidad, ninguno se reproducirá.
4. En la simulación, la mutación en un punto, da como resultado un conjunto de datos que representan una curva más suave.
5. No existen cambios relevantes utilizar el elitismo o al no hacerlo.
6. Al exigir la mutación en cada individuo e iteración de la población, se observará un decrecimiento, debido a una baja tasa de natalidad.
7. Al utilizar el método de selección del torneo, la población mostrará una alta tasa de natalidad.
8. En los algoritmos genéticos el operador de mayor importancia es el operador de selección, ya que éste permite una mejor convergencia en

menor tiempo de ejecución si es bien implantado; pero puede suceder lo contrario si se lo implanta mal.

9. En los algoritmos genéticos la función de adaptación es la misma función que se desea optimizar; pero puede utilizarse algún conocimiento matemático para optimizar el tiempo de ejecución.
10. En los algoritmos genéticos el crossover se encarga de buscar mejores individuos mediante el intercambio de bits y la mutación permite la diversificación de la población para evitar la convergencia prematura; pero el abuso de la mutación provoca una búsqueda aleatoria.
11. Al utilizar un mayor número de individuos se puede abarcar un mayor número de puntos discretos en el intervalo de búsqueda, y por lo tanto se obtendrá estimadores más cercanos a los que se obtienen mediante métodos convencionales o búsquedas heurísticas.
12. Los modelos poblacionales tienen crecimiento que puede modelarse bastante bien mediante el modelo exponencial y el modelo logístico; pero un modelo lineal es muy malo para predecir la tendencia de los mismos.
13. Los estimadores de β_0 y β_1 de los modelos exponenciales y logísticos que se obtienen mediante los algoritmos genéticos son casi los mismos que se obtienen mediante el método convencional de los mínimos cuadrados, siempre que se tome en cuenta el intervalo de búsqueda de estos estimadores.

14. La selección de un modelo que explique la regresión de la mejor manera mediante algoritmos genéticos es una de las principales utilidades que estos algoritmos presentan, ya que cuando existe un gran número de posibles modelos, los algoritmos genéticos pueden encontrar un modelo de gran explicación en menor tiempo que en búsquedas heurísticas.
15. El tiempo de ejecución de los algoritmos genéticos crece exponencialmente y es directamente proporcional al de acuerdo al número de individuos de la población o nodos de la búsqueda, así como al número de iteraciones.
16. Considero que el mejor uso que puede dársele a los algoritmos genéticos aplicándolos a los modelos poblacionales es en la selección de un mejor modelo que dependa de varias variables.

RECOMENDACIONES

1. La selección debe realizarse utilizando como dominio sólo los individuos con mejores aptitudes para obtener una mejor convergencia en menor tiempo. El elitismo también es muy bueno para realizar la selección porque garantiza que se tomarán únicamente los mejores individuos.
2. La función de adaptación debe ser lo más sencilla posible, ya que cuando depende del número de datos, así como en los modelos poblacionales, el tiempo de ejecución aumenta excesivamente.
3. Para la obtención de β_0 y β_1 en los modelos poblacionales, debe utilizarse una probabilidad de crossover mayor a 0.80, para que pueda convergerse a una respuesta adecuada. También debe usarse una probabilidad de mutación que se encuentre entre 0.10 y 0.40 para evitar la convergencia a un punto mínimo local.
4. El tamaño del intervalo de búsqueda y la precisión deseada de los resultados debe ser directamente proporcional al número de nodos o número de individuos que se utilizarán.
5. Antes de escoger y decidir si un modelo de crecimiento poblacional es logístico o exponencial, debe observarse la tendencia de los datos de la población.

6. Los algoritmos genéticos aplicados a los modelos poblacionales deben programarse de la forma más óptima y eficientemente posible, utilizando listas enlazadas para almacenar todos los tipos de datos, y debe minimizarse el acceso a archivos de texto o a bases de datos.
7. Sería apropiado investigar un poco más acerca de los algoritmos genéticos aplicándolos en diferentes ámbitos de la estadística, así como a la investigación de operaciones.
8. Además debe investigarse más sobre los autómatas celulares, ya que es necesario tratar ciertas aplicaciones de la vida artificial que permiten experimentar mediante la simulación.

BIBLIOGRAFIA

- Goldberg, D.E. (1989) "Genetic Algorithms in Search, Optimization and Machine Learning", Reading. Mass.: Addison- Wesley.
- Holland, J. (1975) "Adaptation in Natural and Artificial Systems", The University of Michigan Press, Ann Arbor.
- Dawkins, R. (1982) "The Extended Phenotype: The Gene as a Unit of Selection", Oxford Univ. Press, Oxford.
- Obitko, M. (1998) "Introduction to Genetic Algorithms", Alemania, University of Applied Sciences.
<http://cs.felk.cvut.cz/~xobitko/ga/>
- Herrán , M. (1995) "Vida Artificial", España, El grupo Gaia.
<http://www.geocities.com/SiliconValley/Vista/7491/>
- Contreras, J. (1996) "Operadores genéticos", Venezuela,
<http://www.iamnet.com/users/jcontre/genetic/operadores.htm>
- Larrañaga, P. (2000) "Introducción a los algoritmos genéticos",
<http://www.geocities.com/CapeCanaveral/9802/3d5ca000.htm>
- Ferrer, M. (1997) "Población y recursos naturales", Universidad de navarra, Departamento de humanidades biomedicas
<http://www.eumed.net/cursecon/2/index.htm>
- "Población, Conceptos Básicos"
<http://www.lafacu.com/>
- Preciado, J. (1999) "Futuro",
<http://www.pagina12.com.ar/1999/futuro.htm>
- "La ecología y los insectos", Perú, Universidad de Lamolina
<http://tarwi.lamolina.edu.pe/~acg/ç>
- Cordón J. (1999) "La ecuación logística, modelo poblacional", Dpto. de Matemáticas, Estadística y Computación (MATESCO)
<http://193.144.183.17/~cordon/doctor98/proyecto/logistic.htm>
- "Redes neuronales artificiales"
<http://www.gc.ssr.upm.es/inves/neural/ann2/introduc/intro.htm>

ANEXOS

Anexo 1: Código fuente del Algoritmo Genético Básico

Archivo AG1Dlg.cpp

Programa Principal

```
#include "AG1.h"
#include "AG1Dlg.h"
#include "math.h"
#include "stdio.h"
#include "stdlib.h"
#include "process.h"
#include "Ag.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

genetico G;

class CAboutDlg : public CDialog {
public:
    CAboutDlg();
   //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
   //}}AFX_DATA
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CAboutDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);
   //}}AFX_VIRTUAL
    // Implementation
protected:
   //{{AFX_MSG(CAboutDlg)
   //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD) {
   //{{AFX_DATA_INIT(CAboutDlg)
   //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX) {
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CAboutDlg)
```

```

        //}}AFX_DATA_MAP
    }

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

IMPLEMENT_DYNAMIC(CAG1Dlg, CDialog);

CAG1Dlg::CAG1Dlg(CWnd* pParent /*=NULL*/)
    : CDialog(CAG1Dlg::IDD, pParent) {
    //{{AFX_DATA_INIT(CAG1Dlg)
        m_a = -1.0;
        m_b = 1.0;
        m_iter = 50;
        m_n = 64;
        m_pcruce = 0.90;
        m_pmutar = 0.01;
        m_mejor = 0.0;
        m_mejorada = 0.0;
        m_pcon = 0.0;
        m_promada = 0.0;
        m_ittertot = 0;
    //}}AFX_DATA_INIT
}

CAG1Dlg::~CAG1Dlg() {}

void CAG1Dlg::DoDataExchange(CDataExchange* pDX) {
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAG1Dlg)
        DDX_Control(pDX, IDOK, m_ok);
        DDX_Text(pDX, IDC_a, m_a);
        DDX_Text(pDX, IDC_b, m_b);
        DDX_Text(pDX, IDC_iter, m_iter);
        DDX_Text(pDX, IDC_n, m_n);
        DDX_Text(pDX, IDC_pcruce, m_pcruce);
        DDV_MinMaxDouble(pDX, m_pcruce, 0., 1.);
        DDX_Text(pDX, IDC_pmutar, m_pmutar);
        DDV_MinMaxDouble(pDX, m_pmutar, 0., 1.);
        DDX_Text(pDX, IDC_mejor, m_mejor);
        DDX_Text(pDX, IDC_mejorada, m_mejorada);
        DDX_Text(pDX, IDC_pcon, m_pcon);
        DDV_MinMaxDouble(pDX, m_pcon, 0., 1.);
        DDX_Text(pDX, IDC_promada, m_promada);
        DDX_Text(pDX, IDC_ittertot, m_ittertot);
    //}}AFX_DATA_MAP
}

```

```

        //}}AFX_DATA_MAP
    }

BEGIN_MESSAGE_MAP(CAG1Dlg, CDialog)
    ON_WM_QUERYNEWPALETTE()
    ON_WM_PALETTECHANGED()
    ON_WM_CREATE()
    //{{AFX_MSG_MAP(CAG1Dlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_WM_CLOSE()
    ON_BN_CLICKED(IDC_BUTTON1, OnButton1)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CAG1Dlg::OnInitDialog() {
    CDialog::OnInitDialog();
    ASSERT((IDM_ABOUTBOX & 0xFFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL){
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty()) {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);        // Set small icon
    return TRUE;
}

void CAG1Dlg::OnSysCommand(UINT nID, LPARAM lParam) {
    if ((nID & 0xFFFF0) == IDM_ABOUTBOX) {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else {CDialog::OnSysCommand(nID, lParam);}
}

void CAG1Dlg::OnPaint() {
    if (!IsIconic()) {
        CPaintDC dc(this); // device context for painting
        SendMessage(WM_ICONERASEBKGND, (LPARAM)
dc.GetSafeHdc(), 0);
        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
    }
}

```

```

        int cylcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxlcon + 1) / 2;
        int y = (rect.Height() - cylcon + 1) / 2;
        // Draw the icon
        dc.DrawIcon(x, y, m_hlcon);
    }
    else {CDialog::OnPaint();}
}
HCURSOR CAG1Dlg::OnQueryDragIcon(){return (HCURSOR) m_hlcon;}

void CAG1Dlg::OnClose() {if (CanExit()) CDialog::OnClose();}

void CAG1Dlg::OnOK() {
    UpdateData(TRUE);
    if
(m_a<m_b&&m_pcruce>=0&&m_pcruce<=1&&m_pmutar>=0&&m_pmutar<=1&&m_
pcon>=0&&m_pcon<=1&&m_n>=4&&m_n%2==0&&m_iter>0) {
        G.Crear();
        m_ittertot=G.proceso(m_a,m_b,m_pcruce,m_pmutar,m_pcon,m_n,m_iter,1);
        G.halla_resultados(m_n,0);
        m_mejor=G._mejor();
        m_mejorada=G._mejorada();
        m_promada=G._promada();
        UpdateData(FALSE);
        G.Destruir();
    }//if
    else MessageBox("Por favor, Ingrese correctamente todos los datos...");
}

void CAG1Dlg::OnCancel() {if (CanExit()) CDialog::OnCancel();}

BOOL CAG1Dlg::CanExit(){
    if (m_pAutoProxy != NULL){
        ShowWindow(SW_HIDE);
        return FALSE;
    }
    return TRUE;
}

BOOL CAG1Dlg::OnQueryNewPalette() {
    if (m_pPalette == NULL) return FALSE;
    {
        CClientDC dc(this);
        CPalette* pOldPalette = dc.SelectPalette(m_pPalette,
            GetCurrentMessage()->message ==
WM_PALETTECHANGED);
    }
}

```



```

        UINT nChanged = dc.RealizePalette();
        dc.SelectPalette(pOldPalette, TRUE);
        if (nChanged == 0) return FALSE;
    }
    Invalidate();
    return TRUE;
}

void CAG1Dlg::OnPaletteChanged(CWnd* pFocusWnd) {
    if (pFocusWnd == this || IsChild(pFocusWnd)) return;
    OnQueryNewPalette();
}

CPalette* CAG1Dlg::SetPalette(CPalette* pPalette) {
    CPalette* pOldPalette = m_pPalette;
    m_pPalette = pPalette;
    OnQueryNewPalette();
    return pOldPalette;
}

int CAG1Dlg::OnCreate(LPCREATESTRUCT lpCreateStruct) {
    if (CDialog::OnCreate(lpCreateStruct) == -1) return -1;
    m_pPalette = NULL;
    return 0;
}

void CAG1Dlg::OnButton1() {
    CAboutDlg abo;
    abo.DoModal();
}

```

Archivo de Encabezado AG.h

Clase genético

```
#define MAXLON 20
class genetico {
private:
    struct cromosoma { //Estructura que contiene los cromosomas
        int papa[MAXLON]; //cromosoma original
        int sele[MAXLON]; //cromosoma seleccionado
        long b10;          //genotipo del cromosoma
        double gen;        //genotipo en el intervalo [a,b]
        double ada;        //evaluando en la función de adaptacion
        double dis;        //función relativa del cromosoma
        double acu;        //funcion acumulada
    };
    struct cromosoma2 { //Lista de cromosomas
        struct cromosoma nodo; //campo de datos
        struct cromosoma2 *sig; //puntero a siguiente
    } *lista, *aux;
    struct aleatorio {
        double alea;
        long posicion;
        struct aleatorio *sig;
    } *M1;
    double U;          //Uniforme U(0,1)
    int pos;            //Posicion 1 para cruce y mutacion
    int pos2;           //Posicion 2 para cruce y mutacion
    int LONGEN;         //Longitud de los cromosomas
    double mejorada;    //Mejor disposicion
    double promada;    //Disposicion promedio
    double mejor;       //Valor al que se converge
    double suma;        //Suma de funcion de adaptacion
    long itertot;       //Número de iteraciones que se realizaron
public:
    void Crear(void);    //Constructor
    void Destruir(void); //Destructor

    //Trabajan con el grupo de cromosomas
    void genera_inicial(double,double,int); //genera la poblacion inicial
    long proceso(double,double,double,double,double, long, long, int); //AGBAS
    void selecciona_padres(int); //selecciona los padres
    void ordena_padres(double,double,int); //ordena los padres
    void ordena_hijos(void); //ordena los hijos seleccionados por la adaptacion
    void crossover(int,int,double,int); //realiza el cruce
    void mutacion(int,int,double,int); //realiza la mutacion
    void remplazo(void); //realiza el remplazo de nuevos padres
    int termina(double,int); //verifica si termina el algoritmo
```

```

void halla_resultados(int,double);//halla los resultados
double sumatoria(void);//halla la suma de las funciones de aptitud
void disposicion(void);//halla la disposicion del grupo
void acumulada(void);//halla la disposicion acumulada
double adaptacion(double);//funcion de adaptacion
long base10(int *); //valor decimal del cromosoma
double genotipo(double,double,long,int);
int mejor_ada(double,double,double);
//Entregan Resultados
double _mejor(void) {return mejor;}
double _mejorada(void) {return mejorada;}
double _promada(void) {return promada;}
//Trabajan cada cromosoma individual
double uniforme(void);//genera un número aleatorio uniforme(0,1)
int random(int);//genera uniforme discreto(0,x);
long ubica(double);//ubica de acuerdo a la adaptacion
//Funciones para crossover
//crossover1.-Cruce en un punto
//crossover2.-Cruce en dos punto
//crossoverU.-Cruce Uniforme en todos los punto
void crossover1(cromosoma2 *,double);
void crossover2(cromosoma2 *,double);
void crossoverU(cromosoma2 *,double);
//Funciones para mutacion
//mutacion1.-Mutacion en un punto, cambiando de valor
//mutacion2.-Mutacion en un punto, no siempre cambia de valor
//mutacionU.-Mutacion Uniforme en todos los punto
void mutacion1(cromosoma2 *,double);
void mutacion2(cromosoma2 *,double);
void mutacionU(cromosoma2 *,double);
void sacar(void);
};

void genetico::sacar(void) {
struct cromosoma2 *temp;
if (lista!=NULL) {
temp=lista;
lista=lista->sig;
delete temp;
} //if
}

void genetico::Destruir(void){
aux=lista;
while(lista!=NULL) sacar();
}

void genetico::Crear(void) {

```



```

    mejorada=0.0;promada=0.0;
    suma=0.0;mejor=0.0;
    lista=NULL;aux=NULL;
}

```

```

long genetico::proceso(double a,double b,double Pcruce, double Pmuta,double
Pconv,long n,long iter,int elitismo) {
long cont=1;
genera_inicial(a,b,n);
do {
    selecciona_padres(n);
    ordena_padres(a,b,n);
    crossover(1,n,Pcruce,elitismo);
    mutacion(1,n,Pmuta,elitismo);
    remplazo();
    cont++;
} while (!termina(Pconv,n)&&cont<iter);
return cont;
}

```

```

void genetico::genera_inicial(double a,double b,int n) {
int i,k;
LONGEN=0;
while(n>pow(2,LONGEN)) LONGEN++;
if (pow(2,LONGEN)>n) LONGEN--;
lista=(cromosoma2 *)malloc(sizeof(cromosoma2));
for(i=0;i<LONGEN;i++) lista->nodo.papa[i]=random(2);
for(i=LONGEN;i<MAXLON;i++) lista->nodo.papa[i]=0;
lista->nodo.b10=base10(lista->nodo.papa);
lista->nodo.gen=genotipo(a,b,lista->nodo.b10,n);
lista->nodo.ada=adaptacion(lista->nodo.gen);
lista->nodo.dis=0;
lista->sig=NULL;
aux=lista;
for(k=1;k<n;k++) {
    aux->sig=(cromosoma2 *)malloc(sizeof(cromosoma2));
    aux=aux->sig;
    for(i=0;i<LONGEN;i++) aux->nodo.papa[i]=random(2);
    for(i=LONGEN;i<MAXLON;i++) lista->nodo.papa[i]=0;
    aux->nodo.b10=base10(aux->nodo.papa);
    aux->nodo.gen=genotipo(a,b,aux->nodo.b10,n);
    aux->nodo.ada=adaptacion(aux->nodo.gen);
    aux->nodo.dis=0;
    aux->sig=NULL;
} //for
suma=sumatoria();
disposicion();
acumulada();
}

```

```
}
```

```
void genetico::selecciona_padres(int n) {  
    int i;  
    struct aleatorio *temp;  
    M1=(aleatorio *)malloc(sizeof(aleatorio));  
    M1->sig=NULL;  
    M1->alea=uniforme();  
    M1->posicion=ubica(M1->alea);  
    temp=M1;  
    for (i=0;i<n-1;i++) {  
        temp->sig=(aleatorio *)malloc(sizeof(aleatorio));  
        temp=temp->sig;  
        temp->sig=NULL;  
        temp->alea=uniforme();  
        temp->posicion=ubica(temp->alea);  
    }  
}
```

```
void genetico::ordena_padres(double a,double b,int n) {  
    cromosoma2 *temp;  
    int j=0,k;  
    temp=lista;  
    aleatorio *temp2;  
    temp2=M1;  
    while (temp2!=NULL) {  
        aux=lista;  
        for (j=0;j<temp2->posicion;j++) aux=aux->sig;  
        if (aux!=NULL) {  
            for (k=0;k<LONGEN;k++) temp->nodo.sele[k]=aux->nodo.papa[k];  
            temp->nodo.b10=base10(temp->nodo.sele);  
            temp->nodo.gen=genotipo(a,b,temp->nodo.b10,n);  
            temp->nodo.ada=adaptacion(temp->nodo.gen);  
            temp=temp->sig;  
        }  
        temp2=temp2->sig;  
    }  
    ordena_hijos();  
}
```

```
void genetico::ordena_hijos(void) {  
    cromosoma2 *temp;  
    cromosoma2 aux2;  
    aux=lista;  
    while (aux->sig!=NULL) {  
        temp=aux->sig;  
        while (temp!=NULL) {  
            if (temp->nodo.ada<aux->nodo.ada) {
```

```

        aux2=temp->nodo;
        temp->nodo=aux->nodo;
        aux->nodo=aux2;
    }//if
    temp=temp->sig;
} //while
aux=aux->sig;
} //while
}

void genetico::crossover(int x,int n,double P,int elitismo) {
int i;
    aux=lista;
    if (elitismo==1) {
        for (i=0;i<n/2||i%2==1;i++) aux=aux->sig;
    } //if
    while (aux!=NULL) {
        if (x==1) crossover1(aux,P);
        else if (x==2) crossover2(aux,P);
        else crossoverU(aux,P);
        aux=aux->sig->sig;
    } //while
}

void genetico::mutacion(int x,int n,double P,int elitismo) {
int i;
    aux=lista;
    if (elitismo==1) {
        for (i=0;i<n/2||i%2==1;i++) aux=aux->sig;
    } //if
    while (aux!=NULL) {
        if (x==1) mutacion1(aux,P);
        else if (x==2) mutacion2(aux,P);
        else mutacionU(aux,P);
        aux=aux->sig;
    } //while
}

void genetico::reemplazo(void) {
    suma=sumatoria();
    disposicion();
    acumulada();
}

int genetico::termina(double conv,int n) {
int cont=1;
    aux=lista;
    while (aux->sig!=NULL) {
        if (aux->nodo.gen==aux->sig->nodo.gen) cont++;
    }
}

```

```

    aux=aux->sig;
} //while
if ((double)(cont/n)>=conv) return 1;
return 0;
}

```

```

double genetico::sumatoria(void) {
    suma=0.0;
    aux=lista;
    while (aux!=NULL) {
        suma+=aux->nodo.ada;
        aux=aux->sig;
    } //while
    return suma;
}

```

```

void genetico::disposicion(void) {
    aux=lista;
    while (aux!=NULL) {
        aux->nodo.dis=(double)(aux->nodo.ada/suma);
        aux=aux->sig;
    } //while
}

```

```

void genetico::acumulada(void) {
    double llevo;
    aux=lista;
    aux->nodo.acu=aux->nodo.dis;
    llevo=aux->nodo.dis;
    while (aux->sig!=NULL) {
        aux=aux->sig;
        aux->nodo.acu=aux->nodo.dis+llevo;
        llevo+=aux->nodo.dis;
    } //while
}

```

```

double genetico::adaptacion(double x) {
    return (double)exp(-x*x/2)/sqrt(2*3.14159);
}
long genetico::base10(int A[]) {
    int j;
    float suma=(float)A[0];
    for (j=1;j<LONGEN;j++) {
        suma*=2;
        suma+=(float)(A[j]);
    } //for
    return (long)suma;
}

```

```
double genetico::genotipo(double a,double b,long x,int n) {
    double y;
    y=(double)(b-a)/n;
    return (double)a+x*y;
}
```

```
int genetico::mejor_ada(double x,double y,double z) {
    if ((x-z)>(y-z)) return 1;
    return 0;
}
```

```
double genetico::uniforme(void) {
    return (double)rand()/RAND_MAX;
}
```

```
int genetico::random(int x) {
    double U;
    U=(double)rand()/RAND_MAX;
    return (int)floor(U*x);
}
```

```
long genetico::ubica(double auxi) {
    long cont=0;
    aux=lista;
    while (aux!=NULL) {
        if (auxi<=aux->nodo.acu) return cont;
        aux=aux->sig;
        cont++;
    }//while
    return -1;
}
```

```
void genetico::crossover1(cromosoma2 *temp,double P){
    int i;
    U=uniforme();
    if (P>=U) {
        pos=1+random(LONGEN-1);
        for (i=0;i<pos;i++) {
            temp->nodo.papa[i]=temp->nodo.sele[i];
            temp->sig->nodo.papa[i]=temp->sig->nodo.sele[i];
        }//for
        for (i=pos;i<LONGEN;i++) {
            temp->nodo.papa[i]=temp->sig->nodo.sele[i];
            temp->sig->nodo.papa[i]=temp->nodo.sele[i];
        }//for
    }//if
}
```



```

else {
    for (i=0;i<LONGEN;i++) {
        temp->nodo.papa[i]=temp->nodo.sele[i];
        temp->sig->nodo.papa[i]=temp->sig->nodo.sele[i];
    }//for
} //else
}

```

```

void genetico::crossover2(cromosoma2 *temp,double P){
    int i;
    U=uniforme();
    if (P>=U) {
        do {
            pos=1+random(LONGEN-1);
            pos2=pos+random(LONGEN-pos);
        } while (pos>=pos2);
        for (i=0;i<pos;i++) {
            temp->nodo.papa[i]=temp->nodo.sele[i];
            temp->sig->nodo.papa[i]=temp->sig->nodo.sele[i];
        }//for
        for (i=pos;i<pos2;i++) {
            temp->nodo.papa[i]=temp->sig->nodo.sele[i];
            temp->sig->nodo.papa[i]=temp->nodo.sele[i];
        }//for
        for (i=pos2;i<LONGEN;i++) {
            temp->nodo.papa[i]=temp->nodo.sele[i];
            temp->sig->nodo.papa[i]=temp->sig->nodo.sele[i];
        }//for
    } //if
    else {
        for (i=0;i<LONGEN;i++) {
            temp->nodo.papa[i]=temp->nodo.sele[i];
            temp->sig->nodo.papa[i]=temp->sig->nodo.sele[i];
        }//for
    } //else
}

```

```

void genetico::crossoverU(cromosoma2 *temp,double P){
    int i;
    U=uniforme();
    if (P>=U) {
        for (i=0;i<LONGEN;i++) {
            pos=random(2);
            if (pos==0) {
                temp->nodo.papa[i]=temp->nodo.sele[i];
                temp->sig->nodo.papa[i]=temp->sig->nodo.sele[i];
            } //if
        } else {

```

```

        temp->nodo.papa[i]=temp->sig->nodo.sele[i];
        temp->sig->nodo.papa[i]=temp->nodo.sele[i];
    }//else
}//for
}//if
else {
    for (i=0;i<LONGEN;i++) {
        temp->nodo.papa[i]=temp->nodo.sele[i];
        temp->sig->nodo.papa[i]=temp->sig->nodo.sele[i];
    }//for
}//else
}

void genetico::mutacion1(cromosoma2 *cro,double P) {
    U=uniforme();
    if (U<=P) {
        pos=1+random(LONGEN-1);
        if (cro->nodo.papa[pos]==0) cro->nodo.papa[pos]=1;
        else cro->nodo.papa[pos]=0;
    }//if
}

void genetico::mutacion2(cromosoma2 *cro,double P) {
    U=uniforme();
    if (U<=P) {
        pos=1+random(LONGEN-1);
        cro->nodo.papa[pos]=random(2);
    }//if
}

void genetico::mutacionU(cromosoma2 *cro,double P) {
    int i;
    U=uniforme();
    for (i=0;i<LONGEN;i++) {
        if (U<=P) {
            if (cro->nodo.papa[i]==0) cro->nodo.papa[i]=1;
            else cro->nodo.papa[i]=0;
        }//if
    }//for
}

```

```

void genetico::halla_resultados(int n, double valor) {
    promada=0.0;
    aux=lista;
    mejorada=aux->nodo.ada;
    mejor=aux->nodo.gen;
    while (aux!=NULL) {
        promada+=aux->nodo.ada;
        if (mejor_ada(aux->nodo.ada, mejorada, valor)) {
            mejorada=aux->nodo.ada;
            mejor=aux->nodo.gen;
        }//if
        aux=aux->sig;
    }//while
    promada=(double)(promada/n);
}

```


Anexo 2: Ejecución del Algoritmo Genético Básico

Algoritmo Genético Básico (Normal Estándar)			
Max. Iteraciones:	50	Tam. Población:	64
Extremo a del intervalo:	.1	Extremo b del intervalo:	1
Prob. Crossover:	0.9	Prob. Mutación:	0.01
Prob. Convergencia:	0		
Max. de Iteraciones:	2	Mejor Aptitud:	0.39894244888760
Mejor:	0	Aptitud Promedio:	0.36159393480743
Acercar de ...		Aceptar	Salir

Para este algoritmo utilicé una función normal estándar, y el proceso del algoritmo genético maximiza la función encontrando en el punto cero (0) el valor de la función máximo de 0.39894244888.

Anexo 3: Código fuente del Problema del vendedor viajero (TSP)

Archivo Parameters.cpp

Parámetros del algoritmo

```
#include "stdafx.h"
#include "tsp.h"
#include "Parameters.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CParameters::CParameters(CWnd* pParent /*=NULL*/)
: CDialog(CParameters::IDD, pParent){
   //{{AFX_DATA_INIT(CParameters)
    m_seed = 0;
    m_tournament = 0;
    m_population = 0;
    m_iterations = 0;
   //}}AFX_DATA_INIT
}

void CParameters::DoDataExchange(CDataExchange* pDX){
    CDialog::DoDataExchange(pDX);
    ///{{AFX_DATA_MAP(CParameters)
    DDX_Text(pDX, IDC_EDIT_SEED, m_seed);
    DDX_Text(pDX, IDC_EDIT_TOURNAMENT, m_tournament);
    DDV_MinMaxInt(pDX, m_tournament, 2, 100);
    DDX_Text(pDX, IDC_EDIT_POPULATION, m_population);
    DDV_MinMaxInt(pDX, m_population, 0, 5000);
    DDX_Text(pDX, IDC_EDIT_ITERATIONS, m_iterations);
    DDV_MinMaxInt(pDX, m_iterations, 0, 1000000);
    ///}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CParameters, CDialog)
    ///{{AFX_MSG_MAP(CParameters)
        // NOTE: the ClassWizard will add message map macros here
    ///}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

Archivo de Encabezado Parameters.h

```
#define
AFX_PARAMETERS_H__E0E9EEF0_69C7_11D2_97C4_0060B05C46FD__INCLU
DED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CParameters : public CDialog {
// Construction
public:
    CParameters(CWnd* pParent = NULL); // standard constructor
// Dialog Data
   //{{AFX_DATA(CParameters)
    enum { IDD = IDD_PARAMETERS };
    long    m_seed;
    int      m_tournament;
    int      m_population;
    int      m_iterations;
    //}}AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CParameters)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
   //{{AFX_MSG(CParameters)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before
// the previous line.
#endif
#ifdef AFX_PARAMETERS_H__E0E9EEF0_69C7_11D2_97C4_0060B05C46FD__INCLUDED_
```


Archivo de Encabezado TspDoc.h

```
#if
defined(AFX_TSPDOC_H__4376A5CA_6848_11D2_97C4_0060B05C46FD__INCL
UDED_)
#define
AFX_TSPDOC_H__4376A5CA_6848_11D2_97C4_0060B05C46FD__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CTspDoc : public CDocument {
protected: // create from serialization only
    CTspDoc();
    DECLARE_DYNCREATE(CTspDoc)
// Attributes
public:
    char m_fileName[100];
// Operations
public:
// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CTspDoc)
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);
    //}AFX_VIRTUAL
// Implementation
public:
    virtual ~CTspDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:
    //{AFX_MSG(CTspDoc)
    // NOTE - the ClassWizard will add and remove member functions
    here.
    // DO NOT EDIT what you see in these blocks of generated code !
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
//{AFX_INSERT_LOCATION}
#endif //
defined(AFX_TSPDOC_H__4376A5CA_6848_11D2_97C4_0060B05C46FD__INCL
UDED_)
```

Archivo de Encabezado TspDoc.h

Programa Principal

```
#include "stdafx.h"
#include "tsp.h"
#include "tspDoc.h"
#include "tspView.h"
#include "Parameters.h"
#include "ParamInfo.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

IMPLEMENT_DYNCREATE(CTspView, CView)

BEGIN_MESSAGE_MAP(CTspView, CView)
   //{{AFX_MSG_MAP(CTspView)
    ON_COMMAND(ID_SETUP_PARAMETERS, OnSetupParameters)
    ON_COMMAND(ID_RUN_START, OnRunStart)
    ON_COMMAND(ID_HELP_PARAMETERDESCRIPTIONS,
OnHelpParameterdescriptions)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

CTspView::CTspView() {
    populationSize = 5000;
    numHood = 10;
    numCities = 0;
    maxIterations = 20000;
    seed = 0;
    running = false;
}

CTspView::~CTspView() {}

BOOL CTspView::PreCreateWindow(CREATESTRUCT& cs) {
    return CView::PreCreateWindow(cs);
}
```

```

void CTspView::OnDraw(CDC* pDC) {
    CTspDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // pDC->FillRect(bounds,
    CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
    if ( running ) {
        drawTour( pDC );
    }
}

#ifdef _DEBUG
void CTspView::AssertValid() const { CView::AssertValid();}

void CTspView::Dump(CDumpContext& dc) const { CView::Dump(dc);}

CTspDoc* CTspView::GetDocument() // non-debug version is inline {
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTspDoc)));
    return (CTspDoc*)m_pDocument;
}
#endif // _DEBUG

void CTspView::drawTour(CDC* pdc) {
    BitType cityBit ;
    int currentCity ;
    int lastCity ;
    int i ;
    char tmp[100];
    CRect bounds( 0, 0, 100 * ENLARGE, 100 * ENLARGE + OFFSET );
    pdc->FillRect(bounds,
    CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
    if ( maxIterations == numIterations ) {
        sprintf( tmp, "Iteración : %d Mejor Ruta: %8.2f Iteraciones
realizadas: %d", numIterations, bestFitness, maxIterations );
    }
    else {
        sprintf( tmp, "Iteración : %d Mejor Ruta: %8.2f", numIterations,
bestFitness );
    }
    pdc->TextOut( 10, 5, tmp );
    lastCity = 0 ;
    cityBit = bestTour[ 0 ];
    for ( currentCity = 0 ; !( cityBit & 1 ) ;
        currentCity++, ( cityBit >= 1 ) );
    cityBit = 1 << currentCity ;
    for( i = 1 ; i < numCities+1 ; i++ ) {
        pdc->MoveTo( cityCoords[ currentCity ].x * ENLARGE, cityCoords[
currentCity ].y * ENLARGE + OFFSET );
        pdc->LineTo( cityCoords[ lastCity ].x * ENLARGE, cityCoords[ lastCity
].y * ENLARGE + OFFSET );
    }
}

```

```

        cityBit = bestTour[ currentCity ] & ~( 1 << lastCity );
        lastCity = currentCity ;
    for ( currentCity = 0 ; !( cityBit & 1 ) ;
        currentCity++, ( cityBit >>= 1 ) );
    }
}

```

```

float CTspView::fitness( PersonType person ) {
    int i, lastCity, currentCity ;
    float fitnessValue ;
    BitType cityBit ;
    fitnessValue = 0 ;
    lastCity = 0 ;
    cityBit = person[ 0 ] ;
    for ( currentCity = 0 ; !( cityBit & 1 ) ;
        currentCity++, ( cityBit >>= 1 ) );
    cityBit = 1 << currentCity ;
    for( i = 1 ; i < numCities+1 ; i++ ) {
        fitnessValue += cityDist[ lastCity ][ currentCity ] ;
        cityBit = person[ currentCity ] & ~( 1 << lastCity ) ;
        lastCity = currentCity ;
        for ( currentCity = 0 ; !( cityBit & 1 ) ;
            currentCity++, ( cityBit >>= 1 ) );
    } //for
    return ( fitnessValue ) ;
}

```

```

void CTspView::createPopulation( WorldType population, PopFitnessType
populationFitness ) {
    int i, j, nextCity, lastCity ;
    bestFitness = 10000000 ;
    for ( i = 0 ; i < populationSize ; i++ ) {
        lastCity = 0 ;
        for ( j = 0 ; j < numCities ; j++ )      population[ i ][ j ] = 0 ;
        for ( j = 0 ; j < numCities - 1 ; j++ ) {
            do {
                nextCity = ( rand() % ( numCities - 1 ) ) + 1 ;
            } while ( population[ i ][ nextCity ] );
            population[ i ][ lastCity ] |= 1 << nextCity ;
            population[ i ][ nextCity ] = 1 << lastCity ;
            lastCity = nextCity ;
        } //for
        population[ i ][ lastCity ] |= 1 ;
        population[ i ][ 0 ] |= 1 << lastCity ;
        populationFitness[ i ] = fitness( population[ i ] ) ;
        if ( populationFitness[ i ] < bestFitness )
            bestFitness = populationFitness[ i ] ;
        bestPerson = i ;
    }
}

```



```

    }//for
}

void CTspView::crossover( PersonType child, PersonType p0, PersonType p1 ) {
    int          nextParentPick, i, newCity, newCity1, newCity2, lastCity;
    int          loops, tempLast;
    BitType      cityBit;
    CityCountType cityCount;
    nextParentPick = 0;
    for ( i = 0; i < numCities; i++ ) {
        child[ i ] = 0;
        cityCount[ i ] = 0;
    }//for
    for ( i = 0; i < numCities; i++ )
        if ( cityBit = ( p0[ i ] & p1[ i ] ) ) {
            for ( newCity = 0; !( cityBit & 1 );
                  newCity++, ( cityBit >>= 1 ) );
            child[ i ] = ( 1 << newCity );
            cityCount[ i ] = -1;
            for ( cityBit >>= 1, newCity++; !( cityBit & 1 ) & ( newCity < numCities );
                  newCity++, ( cityBit >>= 1 ) );
            if ( newCity < numCities ) {
                child[ i ] |= ( 1 << newCity );
                cityCount[ i ] = 2;
            }//if
        }//if
    i = 0;
    tempLast = 0;
    lastCity = 0;
    newCity = 0;
    if ( cityCount[ 0 ] != 0 )
        do {
            i++;
            cityBit = child[ newCity ];
            for ( newCity = 0; !( cityBit & 1 ); newCity++,
                  ( cityBit >>= 1 ) );
            if ( newCity == lastCity )
                for ( cityBit >>= 1, newCity++;
                      !( cityBit & 1 ) & ( newCity < numCities );
                      newCity++, ( cityBit >>= 1 ) );
            if ( i == numCities )
                return;
            lastCity = tempLast;
            tempLast = newCity;
        } while ( cityCount[ newCity ] == 2 );
    if ( newCity == 0 )    cityCount[0] = 1;
    else cityCount[0] = 2;
    lastCity = newCity;
}

```



```

for ( ; i < numCities - 1 ; i++ ) {
    loops = 0 ;
    do {
        loops++ ;
        if ( loops == 3 )
            break ;
        if ( nextParentPick )
            cityBit = p0[ lastCity ] ;
        else
            cityBit = p1[ lastCity ] ;
        nextParentPick = ~nextParentPick ;
        for ( newCity1 = 0 ; !( cityBit & 1 ) ;
            newCity1++, ( cityBit >>= 1 ) ) ;
        for ( cityBit >>= 1, newCity2 = newCity1 + 1 ;
            !( cityBit & 1 ) ; newCity2++, ( cityBit >>= 1 ) ) ;
        if ( rand() % 2 ) {
            if ( cityCount[ newCity1 ] != 2 ) newCity = newCity1 ;
            else newCity = newCity2 ;
        } //if
        else {
            if ( cityCount[ newCity2 ] != 2 ) newCity = newCity2 ;
            else newCity = newCity1 ;
        } //else
    } while ( cityCount[ newCity ] == 2 ) ;
    cityCount[ lastCity ] = abs( cityCount[ lastCity ] ) + 1 ;
    if ( loops == 3 )
        do {
            newCity = ( rand() % ( numCities - 1 ) ) + 1 ;
        } while ( cityCount[ newCity ] == 2 ) ;
        child[ lastCity ] |= ( 1 << newCity ) ;
        child[ newCity ] |= ( 1 << lastCity ) ;
        cityCount[ newCity ] = abs( cityCount[ newCity ] ) + 1 ;
        while ( cityCount[ newCity ] == 2 ) {
            i++ ;
            cityBit = child[ newCity ] ;
            tempLast = newCity ;
            for ( newCity = 0 ; !( cityBit & 1 ) ; newCity++, ( cityBit >>= 1 ) ) ;
            if ( newCity == lastCity )
                for ( cityBit >>= 1, newCity++ ; !( cityBit & 1 ) & ( newCity < numCities ) ;
                    newCity++, ( cityBit >>= 1 ) ) ;
            if ( i == numCities )
                return ;
            if ( newCity == 0 ) {
                child[ 0 ] &= ~( 1 << tempLast ) ;
                cityCount[ tempLast ] = 1 ;
                child[ tempLast ] &= ~1 ;
                i-- ;
                newCity = tempLast ;
            }
        }
}

```

```

break ;
} //if
lastCity = tempLast ;
} //while
lastCity = newCity ;
} //for
child[ lastCity ] |= 1 ;
child[ 0 ] |= 1 << lastCity ;
}

void CTspView::makeChildren( WorldType population, PopFitnessType
populationFitness ) {
    int hood[ NUMHOOD ] ;
    int childPos, i, j, best, temp ;
    for ( i = 0 ; i < numHood ; i++ ) hood[ i ] = rand() % populationSize ;
    for ( i = 0 ; i < numHood-1 ; i++ ) {
        best = i ;
        for ( j = i + 1 ; j < numHood ; j++ )
            if ( populationFitness[ hood[ j ] ] < populationFitness[ hood[ best ] ] ) best = j ;
        if ( best != i ) {
            temp = hood[ i ] ;
            hood[ i ] = hood[ best ] ;
            hood[ best ] = temp ;
        } //if
    } //for
    childPos = hood[ numHood - 1 ] ;
    crossover( (PersonType) population[ childPos ], population[ hood[ 0 ] ], population[
hood[ 1 ] ] ) ;
    populationFitness[ childPos ] = fitness( population[ childPos ] ) ;
    if ( populationFitness[ childPos ] < bestFitness ) {
        bestFitness = populationFitness[ childPos ] ;
        bestPerson = childPos ;
    } //if
    childPos = hood[ numHood - 2 ] ;
    crossover( population[ childPos ], population[ hood[ 1 ] ], population[ hood[ 0 ] ] ) ;
    populationFitness[ childPos ] = fitness( population[ childPos ] ) ;
    if ( populationFitness[ childPos ] < bestFitness ) {
        bestFitness = populationFitness[ childPos ] ;
        bestPerson = childPos ;
    } //if
}

void CTspView::displayCityList( PersonType person ) {
    for ( int i = 0 ; i < NUMCITIES ; i++ ) {
        bestTour[i] = person[i] ;
    } //for
    RedrawWindow();
}

```

```

void CTspView::start() {
    WorldType    population ;
    PopFitnessType    populationFitness ;
    float        lastFitness ;
    if ( strlen( GetDocument()->m_fileName ) == 0 ) {
        AfxMessageBox("You must open a city list file first.");
        return;
    }
    running = true;
    if ( seed==0 ) seed=rand();
    else srand( (unsigned)time( NULL ) );
    openCityList( GetDocument()->m_fileName );
    numIterations = 0 ;
    createPopulation( population, populationFitness );
    displayCityList( population[ bestPerson ] );
    lastFitness = bestFitness ;
    for ( int i = 0; i < maxIterations; i++ ) {
        numIterations++;
        makeChildren( population, populationFitness );
        if ( bestFitness < lastFitness ) {
            lastFitness = bestFitness ;
            displayCityList( population[ bestPerson ] );
        }
    }
    RedrawWindow();
}

void CTspView::OnRunStart() {start();}

void CTspView::OnSetupParameters() {
    CParameters params(this);
    params.m_population = populationSize;
    params.m_tournament = numHood;
    params.m_seed = seed;
    params.m_iterations = maxIterations;
    int result = params.DoModal();
    if (result == IDOK) {
        populationSize = params.m_population ;
        numHood = params.m_tournament ;
        seed = params.m_seed ;
        maxIterations = params.m_iterations;
    }
}

void CTspView::openCityList( char *fileName ) {
    FILE *fp ;
    int i, j ;
    if ((fp = fopen(fileName,"r" )) == NULL) {

```

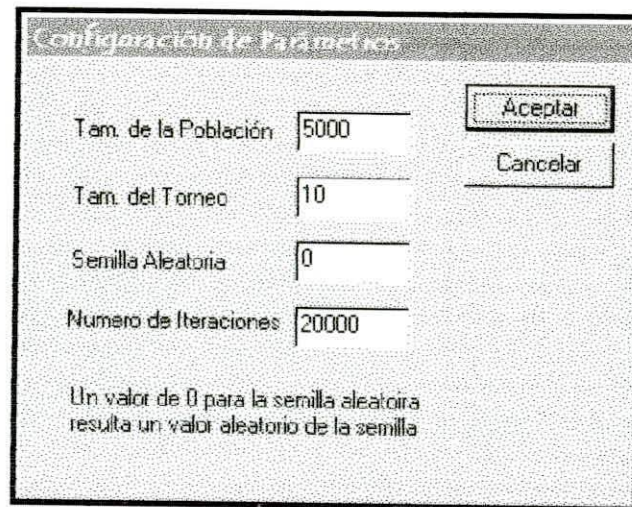
```

        AfxMessageBox( "No se puede abrir el archivo" );
        return;
    } //if
    numCities = 0;
    while ( true ) {
        if ( EOF == fscanf(fp, "%d %d", &cityCoords[ numCities ].x,
&cityCoords[ numCities ].y) ) {
            break;
        } //if
        numCities++;
    } //while
    fclose( fp ) ;
    for ( i = 0 ; i < numCities ; i++ ) {
        for ( j = 0 ; j < numCities ; j++ )
            cityDist[ i ][ j ] = (float) sqrt( (double) ( pow( (double) (
cityCoords[ i ].x - cityCoords[ j ].x ), (double) 2 ) + pow( (double) ( cityCoords[ i ].y -
cityCoords[ j ].y ), (double) 2 ))) ;
    } //for
}

void CTspView::OnHelpParameterdescriptions() {
    CParamInfo info(this);
    info.DoModal();
}

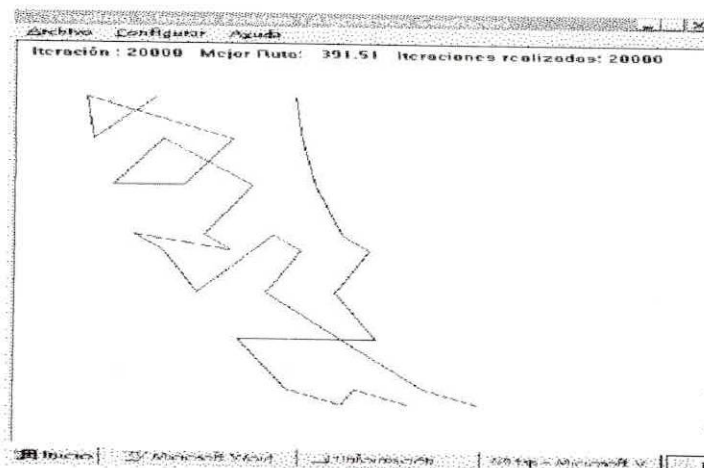
```


Anexo 4: Ejecución del Problema del vendedor viajero (TSP)



The screenshot shows a dialog box titled "Configuración de Parámetros". It contains four input fields with the following values: "Tam. de la Población" is 5000, "Tam. del Torneo" is 10, "Semilla Aleatoria" is 0, and "Numero de Iteraciones" is 20000. To the right of these fields are two buttons: "Aceptar" and "Cancelar". Below the input fields, there is a note: "Un valor de 0 para la semilla aleatoria resulta un valor aleatorio de la semilla".

Esta es la pantalla para el ingreso de parámetros del TSP, aunque en realidad no presenta muchas opciones; pero pide el ingreso de un archivo de texto de las ciudades.



Esta es la pantalla para el ingreso de parámetros del TSP, aunque en realidad no presenta muchas opciones; pero pide el ingreso de un archivo de texto de las ciudades.

Anexo 5: Código fuente de la simulación en tiempo Real.

AgProceDIh.cpp

```
#include "stdafx.h"
#include "AgProce.h"
#include "AgProceDlg.h"
#include "conio.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

class CAboutDlg : public CDialog{
public:
    CAboutDlg();
// Dialog Data
   //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CAboutDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
protected:
   //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD){
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX){
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
    // No messCAgProceDlg handlers

```

```

        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

CAgProceDlg::CAgProceDlg(CWnd* pParent /*=NULL*/):
CDialog(CAgProceDlg::IDD, pParent){
    srand(time(NULL));
    //{{AFX_DATA_INIT(CCAgProceDlgProceDlg)
    m_elitismo = FALSE;
    m_extremos = FALSE;
    m_peor = FALSE;
    m_pcro = 0.9;
    m_pmut = 0.1;
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CAgProceDlg::DoDataExchange(CDataExchange* pDX){
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAgProceDlg)
    DDX_Control(pDX, ID_HELP, m_help);
    DDX_Control(pDX, IDOK, m_ok);
    DDX_Control(pDX, IDCANCEL, m_salir);
    DDX_Check(pDX, IDC_elitismo, m_elitismo);
    DDX_Check(pDX, IDC_extremos, m_extremos);
    DDX_Check(pDX, IDC_peor, m_peor);
    DDX_Text(pDX, IDC_pcro, m_pcro);
    DDV_MinMaxDouble(pDX, m_pcro, 0., 1.);
    DDX_Text(pDX, IDC_pmut, m_pmut);
    DDV_MinMaxDouble(pDX, m_pmut, 0., 1.);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAgProceDlg, CDialog)
    //{{AFX_MSG_MAP(CAgProceDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_DESTROY()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_WM_SHOWWINDOW()
    ON_BN_CLICKED(IDC_unpuntocro, Onunpuntocro)
    ON_BN_CLICKED(IDC_uniformecro, Onuniformecro)
    ON_BN_CLICKED(IDC_unpuntomut, Onunpuntomut)
    ON_BN_CLICKED(IDC_uniformemut, Onuniformemut)
    ON_BN_CLICKED(IDC_ruleta, Onruleta)
    ON_BN_CLICKED(IDC_torneo, Ontorneo)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

BOOL CAgProceDlg::OnInitDialog(){
    CDialog::OnInitDialog();
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL) {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty()) {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
        }
        SetIcon(m_hIcon, TRUE);
        SetIcon(m_hIcon, FALSE);
        return TRUE;
    }
}

void CAgProceDlg::OnSysCommand(UINT nID, LPARAM lParam){
    if ((nID & 0xFFF0) == IDM_ABOUTBOX) {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else CDialog::OnSysCommand(nID, lParam);
}

void CAgProceDlg::OnDestroy(){
    WinHelp(0L, HELP_QUIT);
    CDialog::OnDestroy();
}

void CAgProceDlg::OnPaint() {
    if (IsIconic()) {
        CPaintDC dc(this);
        SendMessage(WM_ICONERASEBKGND, (LPARAM)
dc.GetSafeHdc(), 0);
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;
        dc.DrawIcon(x, y, m_hIcon);
    }
    else CDialog::OnPaint();
}

```



```

HCURSOR CAgProceDlg::OnQueryDragIcon(){return (HCURSOR) m_hIcon;}

void CAgProceDlg::genera_inicial(void) {
    for (int j=0;j<TAMA;j++){
        for (int i=0;i<MAXLON;i++) pap[j][i]=random();
    }//for
    for (int i=25;i<=115;i+=3) {
        for(int j=55;j<=295;j++) cuadro(i,j,RGB(0,0,0));
    }//for
    grafica_reemplazo();
}

void CAgProceDlg::seleccion(BOOL tsele) {
    if (tsele==FALSE) ruleta();
    else torneo();
    grafica_seleccion();
}

long CAgProceDlg::geno(int a[TAMA][MAXLON],int k) {
    long g=pap[k][0];
    for (int i=1;i<MAXLON;i++) {
        g*=2;
        g+=a[k][i];
    }//for
    return g;
}

void CAgProceDlg::ruleta(void) {
    BOOL ff1=TRUE,ff2=TRUE;
    double suma=0;
    double ada[TAMA];
    double U1=(double)rand()/RAND_MAX;
    double U2=(double)rand()/RAND_MAX;
    int k1=0,k2=0;
    for (int i=0;i<TAMA;i++) {
        ada[i]=geno(pap,i);
        suma+=geno(pap,i);
    }//for
    for (i=0;i<TAMA;i++) ada[i]/=suma;
    for (i=1;i<TAMA;i++) ada[i]+=ada[i-1];
    for (i=0;i<TAMA;i++) {
        if (U1<ada[i]&&ff1) {
            k1=i;
            for (int j=0;j<MAXLON;j++) sele[0][j]=pap[i][j];
            ff1=FALSE;
        }//if
        if (U2<ada[i]&&ff2) {

```

```

        k2=i;
        for (int j=0;j<MAXLON;j++) sele[1][j]=pap[i][j];
        ff2=FALSE;
    }//if
}

void CAgProceDlg::torneo(void) {
    int k1=random(TAMA);
    int k2=random(TAMA);
    for (int i=0;i<MAXLON;i++) {
        sele[0][i]=pap[k1][i];
        sele[1][i]=pap[k2][i];
    }//for
}

void CAgProceDlg::crossover(double pcro,BOOL tcro) {
    int i;
    double Cro=(double)rand()/RAND_MAX;
    if
    (Cro<pcro&&(m_elitismo==FALSE||mejor!=geno(sele,0)&&mejor!=geno(sele,1))) {
        if (tcro==FALSE||m_extremos==TRUE) crossover1();
        else crossoverU();
    }//if
    else {
        for (i=0;i<MAXLON;i++) {
            cros[0][i]=sele[0][i];
            cros[1][i]=sele[1][i];
        }//for
    }//else
    grafica_crossover();
}

void CAgProceDlg::crossover1(void) {
    int p,i;
    if (m_extremos==TRUE) p=0;
    else p=random(16);
    for (i=0;i<=p;i++) {
        cros[0][i]=sele[0][i];
        cros[1][i]=sele[1][i];
    }//for
    for (i=p+1;i<MAXLON;i++) {
        cros[0][i]=sele[1][i];
        cros[1][i]=sele[0][i];
    }//for
}

void CAgProceDlg::crossoverU(void) {

```

```

int p,i;
    for (i=0;i<MAXLON;i++) {
        p=random();
        if (p==FALSE) {
            cros[0][i]=sele[0][i];
            cros[1][i]=sele[1][i];
        }//if
        else {
            cros[0][i]=sele[1][i];
            cros[1][i]=sele[0][i];
        }//else
    }//for
}

void CAgProceDlg::mutacion(double pmut,BOOL tmut) {
int i;
double Mut=(double)rand()/RAND_MAX;
    for (i=0;i<MAXLON;i++) {
        muta[0][i]=cros[0][i];
        muta[1][i]=cros[1][i];
    }//for
    if
    (Mut<pmut&&tmut==FALSE&&(m_elitismo==FALSE||mejor!=geno(muta,0)&&mejor!=
    =geno(muta,1))) mutacion1();
    else mutacionU();
    grafica_mutacion();
}

void CAgProceDlg::mutacion1(void) {
int p1=random(16);
int p2=random(16);
    if (geno(muta,0)<promel||m_peor==FALSE) {
        if (muta[0][p1]) muta[0][p1]=FALSE;
        else muta[0][p1]=TRUE;
    }//if
    if (geno(muta,1)<promel||m_peor==FALSE) {
        if (muta[1][p2]) muta[1][p2]=FALSE;
        else muta[1][p2]=TRUE;
    }//if
}

void CAgProceDlg::mutacionU(void) {
int i;
    for (i=0;i<MAXLON;i++) {
        if (random()) muta[0][i]=!muta[0][i];
        if (random()) muta[1][i]=!muta[1][i];
    }//for
}

```

```

void CAgProceDlg::llenar(int k) {
    for (int i=0;i<MAXLON;i++) {
        nue[k][i]=muta[0][i];
        nue[k+1][i]=muta[1][i];
    }//for
    grafica_nueva(k);
}

```

```

void CAgProceDlg::reemplazo(void) {
    for (int j=0;j<TAMA;j++) {
        for (int i=0;i<MAXLON;i++) pap[j][i]=nue[j][i];
    }//for
    for(j=55;j<=295;j++) cuadro(195,j,RGB(0,0,0));
    for(j=55;j<=295;j++) cuadro(205,j,RGB(0,0,0));
    for(j=55;j<=295;j++) cuadro(245,j,RGB(0,0,0));
    for(j=55;j<=295;j++) cuadro(255,j,RGB(0,0,0));
    for(j=55;j<=295;j++) cuadro(295,j,RGB(0,0,0));
    for(j=55;j<=295;j++) cuadro(305,j,RGB(0,0,0));
    for (int i=25;i<=115;i+=3) {
        for(j=55;j<=295;j++) cuadro(i,j,RGB(0,0,0));
    }//for
    for(j=195;j<200;j++){
        cuadro(j,150-(j-195),RGB(250,250,250));
        cuadro(j,150,RGB(250,250,250));
        cuadro(j,150+(j-195),RGB(250,250,250));
    }//for
    for(j=200;j<305;j++) cuadro(j,150,RGB(250,250,250));
    grafica_reemplazo();
    for (i=385;i<=475;i+=3) {
        for(j=55;j<=295;j++) cuadro(i,j,RGB(0,0,0));
    }//for
    for(j=195;j<200;j++){
        cuadro(j,150-(j-195),RGB(0,0,0));
        cuadro(j,150,RGB(0,0,0));
        cuadro(j,150+(j-195),RGB(0,0,0));
    }//for
    for(j=200;j<305;j++) cuadro(j,150,RGB(0,0,0));
}

```

```

void CAgProceDlg::cuadro(int x1,int y1,COLORREF col) {
    CClientDC dc(this);
    for (int i=x1-1;i<=x1+1;i++) {
        for (int j=y1-1;j<=y1+1;j++) dc.SetPixel(i,j,col);
    }//for
}

```



```

void CAgProceDlg::grafica_reemplazo(void) {
    for (int j=0;j<TAMA;j++) {
        for (int i=0;i<MAXLON;i++) {
            if (pap[j][i]==FALSE) cuadro(25+j*6,55+i*6,RGB(250,0,0));
            else cuadro(25+j*6,55+i*6,RGB(0,250,0));
        }    }//for
}

void CAgProceDlg::grafica_seleccion(void) {
    for (int i=0;i<MAXLON;i++) {
        if (sele[0][i]==FALSE) cuadro(195,55+i*6,RGB(250,0,0));
        else cuadro(195,55+i*6,RGB(0,250,0));
        if (sele[1][i]==FALSE) cuadro(205,55+i*6,RGB(250,0,0));
        else cuadro(205,55+i*6,RGB(0,250,0));
    }//for
}

void CAgProceDlg::grafica_crossover(void) {
    for (int i=0;i<MAXLON;i++) {
        if (cros[0][i]==FALSE) cuadro(245,55+i*6,RGB(250,0,0));
        else cuadro(245,55+i*6,RGB(0,250,0));
        if (cros[1][i]==FALSE) cuadro(255,55+i*6,RGB(250,0,0));
        else cuadro(255,55+i*6,RGB(0,250,0));
    }//for
}

void CAgProceDlg::grafica_mutacion(void) {
    for (int i=0;i<MAXLON;i++) {
        if (muta[0][i]==FALSE) cuadro(295,55+i*6,RGB(250,0,0));
        else cuadro(295,55+i*6,RGB(0,250,0));
        if (muta[1][i]==FALSE) cuadro(305,55+i*6,RGB(250,0,0));
        else cuadro(305,55+i*6,RGB(0,250,0));
    }//for
}

void CAgProceDlg::grafica_nueva(int k) {
    for (int j=k;j<k+2;j++) {
        esperar(4000);
        for (int i=0;i<MAXLON;i++) {
            if (nue[j][i]==FALSE) cuadro(385+j*6,55+i*6,RGB(250,0,0));
            else cuadro(385+j*6,55+i*6,RGB(0,250,0));
        }    }//for
}

void CAgProceDlg::Onunpuntocro() {m_tcro=FALSE;}
void CAgProceDlg::Onuniformecro() {m_tcro=TRUE;}
void CAgProceDlg::Onunpuntomut() {m_tmut=FALSE;}
void CAgProceDlg::Onuniformemut() {m_tmut=TRUE;}

```

```

void CAgProceDlg::Onruleta() {m_tsele=FALSE;}
void CAgProceDlg::Ontorneo() {m_tsele=TRUE;}

void CAgProceDlg::OnShowWindow(BOOL bShow, UINT nStatus) {
    CDialog::OnShowWindow(bShow, nStatus);
    CheckRadioButton(IDC_unpuntocro, IDC_uniformecro, IDC_unpuntocro);
    CheckRadioButton(IDC_unpuntomut, IDC_uniformemut, IDC_unpuntomut);
    CheckRadioButton(IDC_ruleta, IDC_torneo, IDC_ruleta);
    m_tcro=FALSE;
    m_tmut=FALSE;
    m_tsele=FALSE;
    bandera=TRUE;
}

void CAgProceDlg::esperar(int x) {
    for (double i=0; i<x*150; i++);
}

void CAgProceDlg::calcula_prome(void) {
    mejor=geno(pap,0);
    for (int i=0; i<TAMA; i++) {
        if (geno(pap,i)>mejor) mejor=geno(pap,i);
    }//for
    prome=0.0;
    for (i=0; i<TAMA; i++) prome+=geno(pap,i);
    prome/=TAMA;
}

void CAgProceDlg::OnOK() {
    UpdateData(TRUE);
    UpdateData(FALSE);
    genera_inicial();
    for (int j=0; j<5; j++) {
        calcula_prome();
        for (int i=0; i<TAMA; i+=2) {
            seleccion(m_tsele);
            esperar(7000);
            crossover(m_pcro, m_tcro);
            esperar(7000);
            mutacion(m_pmut, m_tmut);
            esperar(7000);
            llenar(i);
            esperar(5000);
        }//for
        remplazo();
    }//for
}

```

Archivo de encabezado AgProceDlg.h

```
#include "RoundButton.h"
#include "math.h"
#include "dos.h"
#if
!defined(AFX_AGPROCEDLG_H__B7F16CA7_322C_11D5_AB39_A75187CB9265
__INCLUDED_)
#define
AFX_AGPROCEDLG_H__B7F16CA7_322C_11D5_AB39_A75187CB9265__INCLU
DED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#define MAXLON 40
#define TAMA 16

class CAgProceDlg : public CDialog {
// Construction
public:
    CAgProceDlg(CWnd* pParent = NULL);    // standard constructor
    void genera_inicial(void);
    void crossover(double,BOOL);
    void mutacion(double,BOOL);
    void remplazo(void);
    void seleccion(BOOL);
    void llenar(int);
// Dialog Data
   //{{AFX_DATA(CAgProceDlg)
    enum { IDD = IDD_AGPROCE_DIALOG };
    CRoundButton    m_help;
    CRoundButton    m_ok;
    CRoundButton    m_salir;
    BOOL m_elitismo;
    BOOL m_extremos;
    BOOL m_peor;
    double m_pcro;
    double m_pmut;
    //}}AFX_DATA
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CAgProceDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}}AFX_VIRTUAL
// Implementation
private:
```

```

    BOOL pap[TAMA][MAXLON];
    BOOL sele[2][MAXLON];
    BOOL cros[2][MAXLON];
    BOOL muta[2][MAXLON];
    BOOL nue[TAMA][MAXLON];
    double prome;
    double mejor;
    BOOL bandera;
protected:
    HICON m_hlcon;
    BOOL m_tcro;
    BOOL m_tmut;
    BOOL m_tsele;
    void crossover1(void);
    void crossoverU(void);
    void mutacion1(void);
    void mutacionU(void);
    void cuadro(int,int,COLORREF);
    void grafica_crossover(void);
    void grafica_mutacion(void);
    void grafica_reemplazo(void);
    void grafica_seleccion(void);
    void grafica_nueva(int);
    BOOL random(void) {return ((double)(2.0*rand()/RAND_MAX)<1);}
    int random(int x) {return ((int)floor(x*rand()/RAND_MAX));}
    long random(double x) {return ((long)floor(x*rand()/RAND_MAX));}
    void ruleta(void);
    void torneo(void);
    long geno(int a[][MAXLON],int);
    void esperar(int);
    void calcula_prome(void);
    // Generated message map functions
   //{{AFX_MSG(CAgProceDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnDestroy();
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void Onunpuntocro();
    afx_msg void Onuniformecro();
    afx_msg void Onunpuntomut();
    afx_msg void Onuniformemut();
    virtual void OnOK();
    afx_msg void Onruleta();
    afx_msg void Ontorneo();
   //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

```



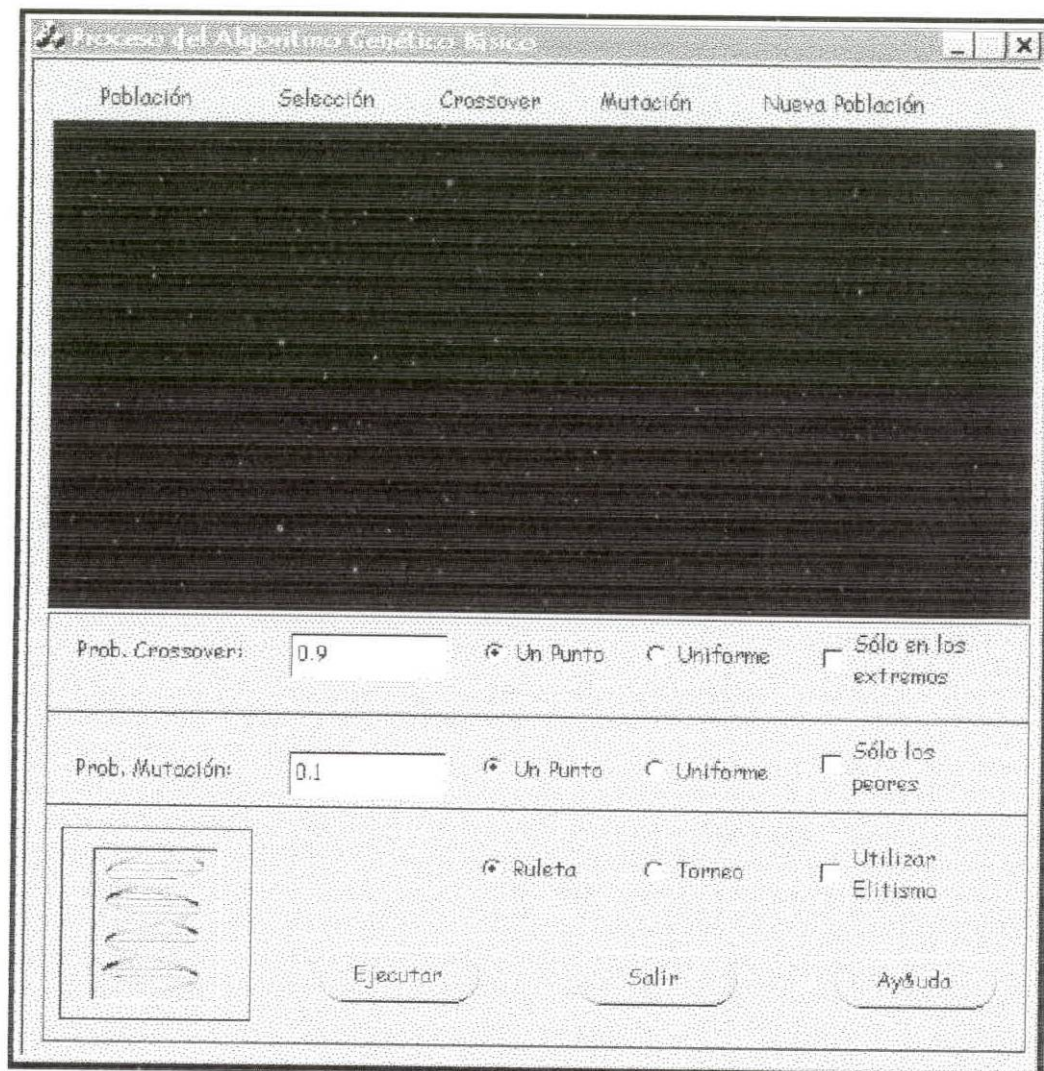
```
};
```

```
//{{AFX_INSERT_LOCATION}}
```

```
#endif //
```

```
!defined(AFX_AGPROCEDLG_H__B7F16CA7_322C_11D5_AB39_A75187CB9265  
__INCLUDED_)
```

Anexo 6: Corrida del Algoritmo genético de Simulación en tiempo Real.



Esta aplicación muestra en una sola pantalla el proceso del algoritmo genético.

Anexo 7: Código fuente del Algoritmo Genético – Modelo Poblacional.

Archivo de Encabezado AgMp.h

```
#if !defined(AFX_AGMP_H__D12FAF47_13B1_11D5_AB39_8BA52D8B6D5D__INCLUDED_)
#define AFX_AGMP_H__D12FAF47_13B1_11D5_AB39_8BA52D8B6D5D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"           // main symbols

class CAgMpApp : public CWinApp{
public:
    CAgMpApp();
// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CAgMpApp)
    public:
        virtual BOOL InitInstance();
    //}AFX_VIRTUAL
// Implementation
    //{AFX_MSG(CAgMpApp)
        // NOTE - the ClassWizard will add and remove member functions here.
        // DO NOT EDIT what you see in these blocks of generated code !
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif // !defined(AFX_AGMP_H__D12FAF47_13B1_11D5_AB39_8BA52D8B6D5D__INCLUDED_)
```

Archivo AgMp.cpp

```
#include "stdafx.h"
#include "AgMp.h"
#include "AgMpDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

BEGIN_MESSAGE_MAP(CAgMpApp, CWinApp)
//{{AFX_MSG_MAP(CAgMpApp)
// NOTE - the ClassWizard will add and remove mapping macros here.
// DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG
ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

CAgMpApp::CAgMpApp(){}

CAgMpApp theApp;

BOOL CAgMpApp::InitInstance(){
    AfxEnableControlContainer();
#ifdef _AFXDLL
    Enable3dControls();
#else
    Enable3dControlsStatic();
#endif
    CAgMpDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK) { }
    else if (nResponse == IDCANCEL) {}
    return FALSE;
}
```

Menú Principal del AgMp

Archivo AgMpDlg.cpp

```
#include "stdafx.h"
#include "AgMp.h"
#include "AgMpDlg.h"
#include "DepTiempo.h"
#include "DepOtras.h"
#include "Simulador.h"
#include "RoundButton.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

class CAboutDlg : public CDialog {
public:
    CAboutDlg();
// Dialog Data
   //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    CRoundButton m_Ok;
   //}}AFX_DATA

    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CAboutDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);
   //}}AFX_VIRTUAL
// Implementation
protected:
   //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD){
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX) {
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    DDX_Control(pDX, IDOK, m_Ok);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

CAgMpDlg::CAgMpDlg(CWnd* pParent /*=NULL*/)
: CDialog(CAgMpDlg::IDD, pParent){
    //{{AFX_DATA_INIT(CAgMpDlg)
```



```

        //}}AFX_DATA_INIT
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    }

void CAgMpDlg::DoDataExchange(CDataExchange* pDX) {
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAgMpDlg)
    DDX_Control(pDX, IDCANCEL, m_salir);
    DDX_Control(pDX, IDC_Simulador, m_simulador);
    DDX_Control(pDX, IDC_Otras, m_otras);
    DDX_Control(pDX, IDC_Tiempo, m_tiempo);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAgMpDlg, CDialog)
    //{{AFX_MSG_MAP(CAgMpDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_DESTROY()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_Simulador, OnSimulador)
    ON_COMMAND(IDM_ABOUTBOX, OnAboutbox)
    ON_BN_CLICKED(IDC_Tiempo, OnTiempo)
    ON_BN_CLICKED(IDC_Otras, OnOtras)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CAgMpDlg::OnInitDialog(){
    CDialog::OnInitDialog();
    ASSERT((IDM_ABOUTBOX & 0xFFFF) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL) {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty()) {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    SetIcon(m_hIcon, TRUE);
    SetIcon(m_hIcon, FALSE);
    return TRUE;
}

void CAgMpDlg::OnSysCommand(UINT nID, LPARAM lParam) {
    if ((nID & 0xFFFF) == IDM_ABOUTBOX) {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else {CDialog::OnSysCommand(nID, lParam);}
}

void CAgMpDlg::OnDestroy(){
    WinHelp(0L, HELP_QUIT);
    CDialog::OnDestroy();
}

void CAgMpDlg::OnPaint() {

```

```

        if (IsIconic()) {
            CPaintDC dc(this);
            SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
            int cxIcon = GetSystemMetrics(SM_CXICON);
            int cyIcon = GetSystemMetrics(SM_CYICON);
            CRect rect;
            GetClientRect(&rect);
            int x = (rect.Width() - cxIcon + 1) / 2;
            int y = (rect.Height() - cyIcon + 1) / 2;
            dc.DrawIcon(x, y, m_hIcon);
        }
    }

    HCURSOR CAgMpDlg::OnQueryDragIcon() {return (HCURSOR) m_hIcon;}

    void CAgMpDlg::OnAboutbox() {
        CAboutDlg Ab;
        Ab.DoModal();
    }

    void CAgMpDlg::OnSimulador() {
        CSimulador Sim;
        srand(NULL);
        Sim.DoModal();
    }

    void CAgMpDlg::OnTiempo() {
        CDepTiempo CDT;
        srand(NULL);
        CDT.DoModal();
    }

    void CAgMpDlg::OnOtras() {
        DepOtras Otr;
        srand(NULL);
        Otr.DoModal();
    }
}

```

Archivo de Encabezado AgMp.h

```
#include "RoundButton.h"
#if !defined(AFX_AGMPDLG_H_D12FAF49_13B1_11D5_AB39_8BA52D8B6D5D__INCLUDED_)
#define AFX_AGMPDLG_H_D12FAF49_13B1_11D5_AB39_8BA52D8B6D5D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CAgMpDlg : public CDialog {
// Construction
public:
    CAgMpDlg(CWnd* pParent = NULL);    // standard constructor
// Dialog Data
   //{{AFX_DATA(CAgMpDlg)
    enum { IDD = IDD_AGMP_DIALOG };
    CRoundButton    m_salir;
    CRoundButton    m_simulador;
    CRoundButton    m_otras;
    CRoundButton    m_tiempo;
    //}}AFX_DATA
// ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CAgMpDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
    protected:
        HICON m_hIcon;
        // Generated message map functions
        {{{AFX_MSG(CAgMpDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnDestroy();
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnSimulador();
        afx_msg void OnAboutbox();
        afx_msg void OnTiempo();
        afx_msg void OnOtras();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
    };

    {{{AFX_INSERT_LOCATION}}
#endif //
#endif(AFX_AGMPDLG_H_D12FAF49_13B1_11D5_AB39_8BA52D8B6D5D__INCLUDED_)
```


Archivo ProgressWnd.cpp

Línea de Progreso

```
#include "stdafx.h"
#include "ProgressWnd.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define IDC_CANCEL 10
#define IDC_TEXT 11
#define IDC_PROGRESS 12

LPCTSTR szSection = _T("Settings");
LPCTSTR szEntryX = _T("X");
LPCTSTR szEntryY = _T("Y");

CProgressWnd::CProgressWnd() {CommonConstruct();}

CProgressWnd::CProgressWnd(CWnd* pParent, LPCTSTR pszTitle, BOOL bSmooth /*= FALSE*/) {
    CommonConstruct();
    m_strTitle = pszTitle;
    Create(pParent, pszTitle, bSmooth);
}

void CProgressWnd::CommonConstruct() {
    m_nNumTextLines = 4;
    m_nPrevPos = 0;
    m_nPrevPercent = 0;
    m_nStep = 1;
    m_nMinValue = 0;
    m_nMaxValue = 100;
    m_strTitle = _T("Línea de Progreso");
    m_strCancelLabel = _T(" Cancelar ");
    m_bCancelled = FALSE;
    m_bModal = FALSE;
    m_bPersistantPosition = TRUE; // saves and restores position automatically
}

CProgressWnd::~CProgressWnd() {DestroyWindow();}

BOOL CProgressWnd::Create(CWnd* pParent, LPCTSTR pszTitle, BOOL bSmooth /*= FALSE*/) {
    BOOL bSuccess;
    CString csClassName = AfxRegisterWndClass(CS_OWNDC|CS_HREDRAW|CS_VREDRAW,
        ::LoadCursor(NULL, IDC_APPSTARTING),
        CBrush(::GetSysColor(COLOR_BTNFACE)));
    NONCLIENTMETRICS ncm;
    ncm.cbSize = sizeof(NONCLIENTMETRICS);
    VERIFY(SystemParametersInfo(SPI_GETNONCLIENTMETRICS, sizeof(NONCLIENTMETRICS),
        &ncm, 0));
    m_font.CreateFontIndirect(&(ncm.lfMessageFont));
    if (!pParent) pParent = AfxGetMainWnd();
    bSuccess = CreateEx(WS_EX_DLGMODALFRAME|WS_EX_TOPMOST, // Extended style
        csClassName, // Classname
        pszTitle, // Title
```

```

        WS_POPUP|WS_BORDER|WS_CAPTION, // style
        0,0, // position - updated soon.
        390,130, // Size - updated soon
        pParent->GetSafeHwnd(), // handle to parent
        0, // No menu
        NULL);
    if (!bSuccess) return FALSE;
    CRect TempRect(0,0,10,10);
    bSuccess = m_Text.Create(_T(""),
        WS_CHILD|WS_VISIBLE|SS_NOPREFIX|SS_LEFTNOWORDWRAP,
        TempRect, this, IDC_TEXT);
    if (!bSuccess) return FALSE;
    DWORD dwProgressStyle = WS_CHILD|WS_VISIBLE;
#ifdef PBS_SMOOTH
    if (bSmooth) dwProgressStyle |= PBS_SMOOTH;
#endif
    bSuccess = m_wndProgress.Create(dwProgressStyle, TempRect, this, IDC_PROGRESS);
    if (!bSuccess) return FALSE;
    bSuccess = m_CancelButton.Create(m_strCancelLabel,
        WS_CHILD|WS_VISIBLE|WS_TABSTOP|BS_PUSHBUTTON,
        TempRect, this, IDC_CANCEL);
    if (!bSuccess) return FALSE;

    m_CancelButton.SetFont(&m_font, TRUE);
    m_Text.SetFont(&m_font, TRUE);
    SetWindowSize(m_nNumTextLines, 390);
    if (m_bPersistantPosition) GetPreviousSettings();
    else CenterWindow();
    Show();
    return TRUE;
}

BOOL CProgressWnd::GoModal(LPCTSTR pszTitle /*= _T("Progress)"/>, BOOL bSmooth /*=FALSE*/)
{
    CWnd *pMainWnd = AfxGetMainWnd();
    if (!IsWindow(m_hWnd) && !Create(pMainWnd, pszTitle, bSmooth)) return FALSE;
    if (pMainWnd) pMainWnd->EnableWindow(FALSE);
    EnableWindow(TRUE);
    m_bModal = TRUE;
    return TRUE;
}

void CProgressWnd::SetWindowSize(int nNumTextLines, int nWindowWidth /*=390*/) {
    int nMargin = 10;
    CSize EdgeSize(::GetSystemMetrics(SM_CXEDGE), ::GetSystemMetrics(SM_CYEDGE));
    CRect TextRect, CancelRect, ProgressRect;
    CSize CancelSize;
    TextRect.SetRect(nMargin,nMargin, nWindowWidth-2*nMargin, 100+2*nMargin);
    CDC* pDC = GetDC();
    if (pDC) {
        CFont* pOldFont = pDC->SelectObject(&m_font);
        CString str = _T("M");
        for (int i = 0; i < nNumTextLines-1; i++) str += _T("\nM");
        pDC->DrawText(str, TextRect, DT_CALCRECT|DT_NOCLIP|DT_NOPREFIX);
        TextRect.right = TextRect.left + nWindowWidth;
        CancelSize = pDC->GetTextExtent(m_strCancelLabel + _T(" ") +
            CString(EdgeSize.cx*4, EdgeSize.cy*3));
        pDC->SelectObject(pOldFont);
        ReleaseDC(pDC);
    }
}

```

```

    }
    CancelRect.SetRect(TextRect.right-CancelSize.cx, TextRect.bottom+nMargin,
        TextRect.right, TextRect.bottom+nMargin + CancelSize.cy);
    ProgressRect.SetRect(TextRect.left, CancelRect.top + EdgeSize.cy,
        CancelRect.left-nMargin, CancelRect.bottom - EdgeSize.cy);
    CSize ClientSize(nMargin + TextRect.Width() + nMargin,
        nMargin + TextRect.Height() + nMargin + CancelRect.Height() + nMargin);
    CRect WndRect, ClientRect;
    GetWindowRect(WndRect); GetClientRect(ClientRect);
    WndRect.right = WndRect.left + WndRect.Width()-ClientRect.Width()+ClientSize.cx;
    WndRect.bottom = WndRect.top + WndRect.Height()-ClientRect.Height()+ClientSize.cy;
    MoveWindow(WndRect);
    m_wndProgress.MoveWindow(ProgressRect);
    m_CancelButton.MoveWindow(CancelRect);
    m_Text.MoveWindow(TextRect);
}

void CProgressWnd::Clear() {
    SetText(_T(""));
    SetPos(0);
    m_bCancelled = FALSE;
    m_nPrevPos = 0;
    if (::IsWindow(GetSafeHwnd()))
        UpdateWindow();
}

void CProgressWnd::Hide() {
    if (!::IsWindow(GetSafeHwnd())) return;
    if (IsWindowVisible()) {
        ShowWindow(SW_HIDE);
        ModifyStyle(WS_VISIBLE, 0);
    }
}

void CProgressWnd::Show() {
    if (!::IsWindow(GetSafeHwnd())) return;
    if (!IsWindowVisible()) {
        ModifyStyle(0, WS_VISIBLE);
        ShowWindow(SW_SHOWNA);
        RedrawWindow(NULL, NULL, RDW_ERASE|RDW_FRAME|RDW_INVALIDATE);
    }
}

void CProgressWnd::SetRange(int nLower, int nUpper, int nStep /* = 1 */) {
    if (!::IsWindow(GetSafeHwnd()))
        return;

#ifdef PBM_SETRANGE32
    ASSERT(-0x7FFFFFFF <= nLower && nLower <= 0x7FFFFFFF);
    ASSERT(-0x7FFFFFFF <= nUpper && nUpper <= 0x7FFFFFFF);
    m_wndProgress.SendMessage(PBM_SETRANGE32, (WPARAM) nLower, (LPARAM) nUpper);
#else
    ASSERT(0 <= nLower && nLower <= 65535);
    ASSERT(0 <= nUpper && nUpper <= 65535);
    m_wndProgress.SetRange(nLower, nUpper);
#endif
    m_nMaxValue = nUpper;
    m_nMinValue = nLower;
    m_nStep = nStep;
}

```



```

    m_wndProgress.SetStep(nStep);
}

int CProgressWnd::OffsetPos(int nPos) {
    if (!::IsWindow(GetSafeHwnd()))
        return m_nPrevPos;
    Show();
    return SetPos(m_nPrevPos + nPos);
}

int CProgressWnd::StepIt() {
    if (!::IsWindow(GetSafeHwnd()))
        return m_nPrevPos;
    Show();
    return SetPos(m_nPrevPos + m_nStep);
}

int CProgressWnd::SetStep(int nStep) {
    int nOldStep = m_nStep;
    m_nStep = nStep;
    if (!::IsWindow(GetSafeHwnd()))
        return nOldStep;
    return m_wndProgress.SetStep(nStep);
}

int CProgressWnd::SetPos(int nPos) {
#ifdef PBM_SETRANGE32
    ASSERT(-0x7FFFFFFF <= nPos && nPos <= 0x7FFFFFFF);
#else
    ASSERT(0 <= nPos && nPos <= 65535);
#endif

    if (!::IsWindow(GetSafeHwnd()))
        return m_nPrevPos;
    Show();
    CString strTitle;
    int nPercentage;
    m_nPrevPos = nPos;
    if (m_nMaxValue > m_nMinValue)
        nPercentage = (nPos*100)/(m_nMaxValue - m_nMinValue);
    else
        nPercentage = 0;
    if (nPercentage != m_nPrevPercent) {
        m_nPrevPercent = nPercentage;
        strTitle.Format(_T("%s [%d%%]"), m_strTitle, nPercentage);
        SetWindowText(strTitle);
    }
    return m_wndProgress.SetPos(nPos);
}

void CProgressWnd::SetText(LPCTSTR fmt, ...) {
    if (!::IsWindow(GetSafeHwnd()))
        return;
    va_list args;
    TCHAR buffer[512];
    va_start(args, fmt);
    _vstprintf(buffer, fmt, args);
    va_end(args);
    m_Text.SetWindowText(buffer);
}

```

```

}

BEGIN_MESSAGE_MAP(CProgressWnd, CWnd)
//{{AFX_MSG_MAP(CProgressWnd)
ON_WM_ERASEBKGD()
//}}AFX_MSG_MAP
ON_BN_CLICKED(IDC_CANCEL, OnCancel)
END_MESSAGE_MAP()

BOOL CProgressWnd::OnEraseBkgnd(CDC* pDC) {
    CBrush backBrush(GetSysColor(COLOR_BTNFACE));
    CBrush* pOldBrush = pDC->SelectObject(&backBrush);
    CRect rect;
    pDC->GetClipBox(&rect);
    pDC->PatBlt(rect.left, rect.top, rect.Width(), rect.Height(), PATCOPY);
    pDC->SelectObject(pOldBrush);
    return TRUE;
}

void CProgressWnd::OnCancel() {
    m_bCancelled = TRUE;
    Hide();
    if (m_bModal) SendMessage(WM_CLOSE);
    CWnd* pWnd = AfxGetMainWnd();
    if (pWnd && !::IsWindow(pWnd->m_hWnd))
        pWnd->SetForegroundWindow();
}

BOOL CProgressWnd::DestroyWindow() {
    if (m_bPersistantPosition) SaveCurrentSettings();
    if (m_bModal) {
        m_bModal = FALSE;
        CWnd* pMainWnd = AfxGetMainWnd();
        if (pMainWnd)
            pMainWnd->EnableWindow(TRUE);
    }
    return CWnd::DestroyWindow();
}

void CProgressWnd::PeekAndPump(BOOL bCancelOnESCkey /*= TRUE*/) {
    if (m_bModal && !::GetFocus() != m_hWnd)
        SetFocus();
    MSG msg;
    while (!m_bCancelled && !::PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE)) {
        if (bCancelOnESCkey && (msg.message == WM_CHAR) && (msg.wParam == VK_ESCAPE))
            OnCancel();
        if (m_bModal && (msg.message == WM_LBUTTONDOWN)) {
            CRect rect;
            m_CancelButton.GetWindowRect(rect);
            if (rect.PtInRect(msg.pt)) OnCancel();
        }
        if (!AfxGetApp()->PumpMessage()) {
            ::PostQuitMessage(0);
            return;
        }
    }
}

```

```

void CProgressWnd::GetPreviousSettings() {
    int x = AfxGetApp()->GetProfileInt(szSection, szEntryX, -1);
    int y = AfxGetApp()->GetProfileInt(szSection, szEntryY, -1);
    if (x >= 0 && x < GetSystemMetrics(SM_CXSCREEN) &&
        y >= 0 && y < GetSystemMetrics(SM_CYSCREEN)) {
        SetWindowPos(NULL, x,y, 0,0, SWP_NOSIZE|SWP_NOZORDER);
    }//if
    else CenterWindow();
}

void CProgressWnd::SaveCurrentSettings() {
    if (!IsWindow(m_hWnd)) return;
    CRect rect;
    GetWindowRect(rect);
    AfxGetApp()->WriteProfileInt(szSection, szEntryX, rect.left);
    AfxGetApp()->WriteProfileInt(szSection, szEntryY, rect.top);
}

```

Archivo de encabezado ProgresWnd.h

```
#ifndef _INCLUDE_PROGRESSWND_H
#define _INCLUDE_PROGRESSWND_H

class CProgressWnd : public CWnd{
// Construction/Destruction
public:
    CProgressWnd();
    CProgressWnd(CWnd* pParent, LPCTSTR pszTitle, BOOL bSmooth = FALSE);
    virtual ~CProgressWnd();
    BOOL Create(CWnd* pParent, LPCTSTR pszTitle, BOOL bSmooth = FALSE);
    BOOL GoModal(LPCTSTR pszTitle = _T("Progress"), BOOL bSmooth = FALSE);
protected:
    void CommonConstruct();
// Operations
public:
    void SetRange(int nLower, int nUpper, int nStep = 1);
    int OffsetPos(int nPos);           // Same as CProgressCtrl
    int StepIt();                     // "
    int SetStep(int nStep);           // "
    int SetPos(int nPos);              // "
    void SetText(LPCTSTR fmt, ...);    // Set text in text area
    void Clear();                     // Clear text, reset bar
    void Hide();                      // Hide window
    void Show();                      // Show window
    BOOL Cancelled() { return m_bCancelled; } // Was "Cancel" hit?
    void SetWindowSize(int nNumTextLines, int nWindowWidth = 390);
    void PeekAndPump(BOOL bCancelOnESCkey = TRUE); // Message pumping for modal operation
// Implementation
protected:
    void GetPreviousSettings();
    void SaveCurrentSettings();
protected:
    BOOL m_bCancelled, m_bModal, m_bPersistantPosition;
    int m_nPrevPos, m_nPrevPercent, m_nStep, m_nMaxValue, m_nMinValue;
    int m_nNumTextLines;
    CStatic m_Text;
    CProgressCtrl m_wndProgress;
    CButton m_CancelButton;
    CString m_strTitle, m_strCancelLabel;
    CFont m_font;
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CProgressWnd)
    public:
        virtual BOOL DestroyWindow();
    //}}AFX_VIRTUAL

// Generated message map functions
protected:
    //{{AFX_MSG(CProgressWnd)
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    //}}AFX_MSG
    afx_msg void OnCancel();
    DECLARE_MESSAGE_MAP()
};
#endif
```


Archivo FileEditCtrl.cpp

Abrir un archivo

```
#include "stdafx.h"
#include "resource.h"
#include "FileEditCtrl.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#if !defined FEC_IDS_ALLFILES
#define FEC_NORESOURCETRINGS so this class knows how to handle these strings
#define FEC_IDS_ALLFILES _T("Modelos Poblacionales")
(*.Agm;*.Agp)|*.Agm;*.Agp|")
#define FEC_IDS_FILEDIALOGTITLE _T("Buscando un archivo")
#define FEC_IDS_SEPERATOR _T(";")
#define FEC_IDS_NOFILE _T("El archivo no fue encontrado.")
#define FEC_IDS_NOTEXIST _T("%s No existe.")
#define FEC_IDS_NOTFILE _T("%s No es un archivo válido.")
#define FEC_IDS_NOTFOLDER _T("%s No es una carpeta válida.")
#define FEC_IDS_OKBUTTON _T("Aceptar")
#endif

#define BTN_UP 0
#define BTN_DOWN 1
#define BTN_DISABLED 2

BOOL IsWindow (CWnd *pWnd) {
    if (!pWnd)
        return (FALSE);
    return ::IsWindow (pWnd->m_hWnd);
}

IMPLEMENT_DYNAMIC (CFileEditCtrl, CEdit)

CFileEditCtrl::CFileEditCtrl(BOOL bAutoDelete /* = FALSE */) {
    m_bAutoDelete = bAutoDelete;
    m_bButtonLeft = FALSE;
    m_bCreatingControl = TRUE;
    m_bFindFolder = FALSE;
    m_bMouseCaptured = FALSE;
    m_bTextChanged = TRUE;
    m_bTrailingSlash = FALSE;
    m_lpstrFiles = NULL;
    m_nButtonState = BTN_UP;
    m_pBROWSEINFO = NULL;
    m_pCFileDialog = NULL;
    m_rcButtonRect.SetRectEmpty();
    m_szCaption.Empty();
    m_szFolder.Empty();
}

CFileEditCtrl::~CFileEditCtrl() {
    if (m_lpstrFiles) delete m_lpstrFiles;
    if (m_pBROWSEINFO) delete m_pBROWSEINFO;
```



```

        if (m_pFileDialog) delete m_pFileDialog;
    }

BEGIN_MESSAGE_MAP(CFileEditCtrl, CEdit)
    ///{{AFX_MSG_MAP(CFileEditCtrl)
    ON_CONTROL_REFLECT_EX(EN_CHANGE, OnChange)
    ON_WM_DESTROY()
    ON_WM_DROPFILES()
    ON_WM_ENABLE()
    ON_WM_KEYDOWN()
    ON_WM_LBUTTONDOWN()
    ON_WM_MOUSEMOVE()
    ON_WM_NCCALCSIZE()
    ON_WM_NCHITTEST()
    ON_WM_NCLBUTTONDOWN()
    ON_WM_NCPAINT()
    ON_WM_SETFOCUS()
    ///}}AFX_MSG_MAP
END_MESSAGE_MAP()

void CFileEditCtrl::ButtonClicked(){
    BOOL bResult = FALSE;
    if (m_bFindFolder)        bResult = FECBrowseForFolder();
    else bResult = FECOpenFile();
    if (bResult && IsWindow(GetParent()))
        GetParent()->PostMessage(WM_COMMAND, MAKEWPARAM(GetDlgCtrlID(),
BN_CLICKED), (LPARAM)GetSafeHwnd());
}

BOOL CFileEditCtrl::Create(DWORD dwFlags, DWORD dwExStyle, LPCTSTR lpszWindowName,
DWORD dwStyle, const RECT& rect, CWnd* pParentWnd, UINT nID) {
    BOOL bResult = CreateEx(dwExStyle, "EDIT", lpszWindowName, dwStyle, rect, pParentWnd,
nID);
    if (bResult){
        bResult = SetFlags(dwFlags);

        SetWindowPos(NULL,0,0,0,0,SWP_FRAMECHANGED|SWP_NOMOVE|SWP_NOSIZE|SWP
_NOZORDER|SWP_NOACTIVATE);
        if (pParentWnd) SetFont (pParentWnd->GetFont());
    }
    return bResult;
}

void CFileEditCtrl::DrawButton(int nButtonState){
    ASSERT (IsWindow(this));
    if (GetStyle() & WS_DISABLED) nButtonState = BTN_DISABLED;
    CWindowDC DC(this);
    CBrush theBrush(GetSysColor(COLOR_3DFACE));
    CBrush *pOldBrush = DC.SelectObject(&theBrush);
    if (nButtonState == BTN_DOWN) {
        CPen thePen(PS_SOLID, 1, GetSysColor(COLOR_3DSHADOW));
        CPen *pOldPen = DC.SelectObject(&thePen);
        DC.Rectangle(&m_rcButtonRect);
        DC.SelectObject(pOldPen);
        thePen.DeleteObject();
        DrawDots (&DC, GetSysColor(COLOR_BTNTEXT), 1);
    }
    else {
        CPen thePen(PS_SOLID, 1, GetSysColor(COLOR_3DFACE));
    }
}

```

```

        CPen *pOldPen = DC.SelectObject(&thePen);
        DC.Rectangle(&m_rcButtonRect);
        DC.SelectObject(pOldPen);
        thePen.DeleteObject();
        DC.DrawEdge(&m_rcButtonRect, EDGE_RAISED, BF_RECT);
        if (nButtonState == BTN_DISABLED) {
            DrawDots (&DC, GetSysColor(COLOR_3DHILIGHT), 1);
            DrawDots (&DC, GetSysColor(COLOR_GRAYTEXT));
        }
        else if (nButtonState == BTN_UP)
            DrawDots (&DC, GetSysColor(COLOR_BTNTEXT));
        else ASSERT (FALSE);
    }
    DC.SelectObject(pOldBrush);
    theBrush.DeleteObject();
    m_nButtonState = nButtonState;
}

void CFileEditCtrl::DrawDots(CDC *pDC, COLORREF CR, int nOffset /* = 0 */){
    int width = m_rcButtonRect.Width();
    div_t divt = div (width, 4);
    int delta = divt.quot;
    int left = m_rcButtonRect.left + width / 2 - delta - (divt.rem ? 0 : 1);
    width = width / 10;
    int top = m_rcButtonRect.Height() / 2 - width / 2 + 1;
    left += nOffset;
    top += nOffset;
    if (width < 2) {
        pDC->SetPixel(left, top, CR);
        left += delta;
        pDC->SetPixel(left, top, CR);
        left += delta;
        pDC->SetPixel(left, top, CR);
    }
    else {
        CPen thePen(PS_SOLID, 1, CR);
        CPen *pOldPen = pDC->SelectObject(&thePen);
        CBrush theBrush(CR);
        CBrush *pOldBrush = pDC->SelectObject(&theBrush);
        pDC->Ellipse(left, top, left + width, top + width);
        left += delta;
        pDC->Ellipse(left, top, left + width, top + width);
        left += delta;
        pDC->Ellipse(left, top, left + width, top + width);
        pDC->SelectObject(pOldBrush);
        theBrush.DeleteObject();
        pDC->SelectObject(pOldPen);
        thePen.DeleteObject();
    }
}

int CALLBACK FECFolderProc(HWND hWnd, UINT nMsg, LPARAM, LPARAM lpData) {
    if (nMsg == BFFM_INITIALIZED)
        ::SendMessage(hWnd, BFFM_SETSELECTION, TRUE, lpData);
    return 0;
}

BOOL CFileEditCtrl::FECBrowseForFolder(){
    BOOL bReturnValue = FALSE;

```

```

    BOOL bProcedure = TRUE;
    BOOL bDisplay = TRUE;
    TCHAR lpstrDisplay[_MAX_PATH];
    if (!m_pBROWSEINFO->pszDisplayName) {
        bDisplay = FALSE;
        m_pBROWSEINFO->pszDisplayName = lpstrDisplay;
    }
    CString szPath;
    LPARAM oldLP = m_pBROWSEINFO->IParam;
    if (!m_pBROWSEINFO->lpfn) {
        bProcedure = FALSE;
        POSITION pos = GetStartPosition();
        if (pos) {
            szPath = GetNextPathName(pos);
            int len = szPath.GetLength() - 1;
            if (len != 2 && szPath[len] == '\\') szPath.Delete(len);
        }
        m_pBROWSEINFO->IParam = (LPARAM)(LPCTSTR)szPath;
        m_pBROWSEINFO->lpfn = FECFolderProc;
    }
    ITEMIDLIST *idl = SHBrowseForFolder(m_pBROWSEINFO);
    if (idl) {
        if (SHGetPathFromIDList(idl, lpstrDisplay)) {
            if (m_bTrailingSlash) {
                int len = _tcslen(lpstrDisplay);
                if (lpstrDisplay[len - 1] != '\\')
                    _tcscat(lpstrDisplay, _T("\\"));
            }
            SetWindowText(lpstrDisplay);
            bReturnValue = TRUE;
        }
        LPMALLOC lpm;
        if (SHGetMalloc(&lpm) == NOERROR) lpm->Free(idl);
    }

    if (!bDisplay) m_pBROWSEINFO->pszDisplayName = NULL;
    if (!bProcedure) {
        m_pBROWSEINFO->lpfn = NULL;
        m_pBROWSEINFO->IParam = oldLP;
    }
    SetFocus();
    return bReturnValue;
}

BOOL CFileEditCtrl::FECOpenFile() {
    BOOL bReturnValue = FALSE;
    BOOL bDirectory = TRUE;
    TCHAR lpstrDirectory[_MAX_PATH] = _T("");
    if (m_pCFileDialog->m_ofn.lpstrInitialDir == NULL) {
        bDirectory = FALSE;
        POSITION pos = GetStartPosition();
        if (pos) _tcscpy(lpstrDirectory, GetNextPathName(pos));
        m_pCFileDialog->m_ofn.lpstrInitialDir = lpstrDirectory;
    }
    if (m_pCFileDialog->DoModal() == IDOK) {
        CString szFileSeperator;
#ifdef FEC_NORESOURCESTRINGS
        szFileSeperator = (CString)FEC_IDS_SEPERATOR;
#endif
    }
}

```



```

#else
    szFileSeperator.LoadString(FEC_IDS_SEPERATOR);
#endif

    ASSERT(_tcslen(szFileSeperator) == 1);
    szFileSeperator += _T(" ");
    CString szPath;
    POSITION pos = m_pCFileDialog->GetStartPosition();
    if (pos) szPath = m_pCFileDialog->GetNextPathName(pos);
    TCHAR lpstrName[_MAX_PATH];
    TCHAR lpstrExt[_MAX_PATH];
    CString szTempPath;
    while (pos)
    {
        szTempPath = m_pCFileDialog->GetNextPathName(pos);
        _tsplitpath(szTempPath, NULL, NULL, lpstrName, lpstrExt);
        szPath += szFileSeperator + lpstrName + lpstrExt;
    }
    while
    SetWindowText(szPath);
    bReturnValue = TRUE;
}
//if
if (!bDirectory) m_pCFileDialog->m_ofn.lpstrInitialDir = NULL;
SetFocus();
return bReturnValue;
}

void CFileEditCtrl::FillBuffers()
{
    ASSERT(IsWindow(this));
    #if defined FEC_NORESOURCESTRINGS
        m_szFolder = FEC_IDS_SEPERATOR;
    #else
        m_szFolder.LoadString(FEC_IDS_SEPERATOR);
    #endif

    TCHAR chSeperator = m_szFolder[0];
    m_szFolder.Empty();
    if (m_lpstrFiles)
    {
        delete m_lpstrFiles;
        m_lpstrFiles = NULL;
    }
    //if
    int len = GetWindowTextLength();
    if (!len) return;
    LPTSTR lpstrWindow = new TCHAR[len + 1];
    GetWindowText(lpstrWindow, len + 1);
    LPTSTR lpstrStart = lpstrWindow;
    LPTSTR lpstrEnd = NULL;
    while (*lpstrStart == chSeperator || _istspace(*lpstrStart))    lpstrStart++;
    if (!*lpstrStart)
    {
        delete lpstrWindow;
        return;
    }
    //if
    lpstrEnd = _tcschr(lpstrStart, chSeperator);
    if (lpstrEnd) *lpstrEnd = 0;
    if (!lpstrEnd || m_bFindFolder || (!m_bFindFolder && !(m_pCFileDialog->m_ofn.Flags & OFN_ALLOWMULTISELECT)))
    {
        m_lpstrFiles = new TCHAR[_MAX_PATH];
        ZeroMemory(m_lpstrFiles, _MAX_PATH);
        _tfullpath(m_lpstrFiles, lpstrStart, _MAX_PATH);
        if (m_bTrailingSlash && m_bFindFolder)
        {
            int len = _tcslen(m_lpstrFiles);
            if (m_lpstrFiles[len - 1] != '\\')

```

```

        _tcscat(m_lpstrFiles, _T("\\"));
    }
    delete lpstrWindow;
    return;
}

_TCHAR Drive[_MAX_DRIVE];
_TCHAR Folder[_MAX_PATH];
_TCHAR File[_MAX_PATH];
_TCHAR Ext[_MAX_PATH];
_tsplitpath(lpstrStart, Drive, Folder, File, Ext);
m_szFolder = (CString)Drive + Folder;
m_lpstrFiles = new TCHAR[Len + 1];
ZeroMemory(m_lpstrFiles, Len + 1);
_tstrcpy(m_lpstrFiles, File);
_tcscat(m_lpstrFiles, Ext);
lpstrStart = lpstrEnd + 1;
LPTSTR pointer = m_lpstrFiles + _tcslen(m_lpstrFiles) + 1;
while (lpstrEnd) {
    while (*lpstrStart == chSeperator || _istspace(*lpstrStart)) lpstrStart++;
    if (!*lpstrStart) break;
    lpstrEnd = _tcschr(lpstrStart, chSeperator);
    if (lpstrEnd) *lpstrEnd = 0;
    _tstrcpy(pointer, lpstrStart);
    pointer += _tcslen(pointer) + 1;
    if (lpstrEnd) lpstrStart = lpstrEnd + 1;
}
delete lpstrWindow;
}

BROWSEINFO* CFileEditCtrl::GetBrowseInfo() const {return m_pBROWSEINFO;}

DWORD CFileEditCtrl::GetFlags() {
    DWORD flags = m_bFindFolder ? FEC_FOLDER : FEC_FILE;
    if (m_bTrailingSlash) flags |= FEC_TRAILINGSLASH;
    if (m_bButtonLeft) flags |= FEC_BUTTONLEFT;
    return flags;
}

CString CFileEditCtrl::GetNextPathName(POSITION &pos){
    ASSERT (pos);
    LPTSTR str = (LPTSTR)pos;
    TCHAR lpstrReturnString[_MAX_PATH];
    CString szTemp;
    if (str[1] == ':') szTemp = str;
    else
        szTemp = m_szFolder + str;
    _tfullpath(lpstrReturnString, szTemp, _MAX_PATH);
    str += _tcslen(str) + 1;
    pos = *str ? (POSITION)str : NULL;
    return (CString)lpstrReturnString;
}

OPENFILENAME* CFileEditCtrl::GetOpenFileName() const{
    if (m_pCFileDialog) return (&m_pCFileDialog->m_ofn);
    return NULL;
}

POSITION CFileEditCtrl::GetStartPosition(){
    if (IsWindow(this) && m_bTextChanged) {

```

```

        FillBuffers();
        m_bTextChanged = FALSE;
    } //if
    return (POSITION)m_lpstrFiles;
}

BOOL CFileEditCtrl::OnChange() {
    m_bTextChanged = TRUE;
    return FALSE;
}

void CFileEditCtrl::OnDestroy() {if (m_bAutoDelete) delete this;}

void CFileEditCtrl::OnDropFiles(HDROP hDropInfo) {
    CString szSeperator;
    #if defined FEC_NORESOURCESTRINGS
        szSeperator = FEC_IDS_SEPERATOR;
    #else
        szSeperator.LoadString(FEC_IDS_SEPERATOR);
    #endif
    ASSERT (_tcslen(szSeperator) == 1);
    szSeperator += _T(" ");
    CString szDroppedFiles;
    TCHAR lpstrDropBuffer[_MAX_PATH];
    UINT nDropCount = DragQueryFile(hDropInfo, 0xffffffff, NULL, 0);
    if (nDropCount && (m_bFindFolder || (!m_bFindFolder && !(m_pCFileDialog->m_ofn.Flags &
    OFN_ALLOWMULTISELECT)))) nDropCount = 1;
    if (nDropCount) {

        DragQueryFile(hDropInfo, 0, lpstrDropBuffer, _MAX_PATH);
        szDroppedFiles = lpstrDropBuffer;
    } //if
    TCHAR lpstrName[_MAX_PATH];
    TCHAR lpstrExt[_MAX_PATH];
    for (UINT x = 1; x < nDropCount; x++) {
        DragQueryFile(hDropInfo, x, lpstrDropBuffer, _MAX_PATH);
        _tsplitpath(lpstrDropBuffer, NULL, NULL, lpstrName, lpstrExt);
        szDroppedFiles += szSeperator + lpstrName + lpstrExt;
    } //for
    DragFinish(hDropInfo);
    SetWindowText(szDroppedFiles);
}

void CFileEditCtrl::OnEnable(BOOL bEnable) {
    CEdit::OnEnable(bEnable);
    DrawButton(bEnable ? BTN_UP : BTN_DISABLED);
}

void CFileEditCtrl::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags) {
    if ((nChar == 0xBE || nChar == VK_DECIMAL) && GetKeyState(VK_CONTROL) < 0)
        ButtonClicked();
    CEdit::OnKeyDown(nChar, nRepCnt, nFlags);
}

void CFileEditCtrl::OnLButtonUp(UINT nFlags, CPoint point) {
    CEdit::OnLButtonUp(nFlags, point);
    if (m_bMouseCaptured) {
        ReleaseCapture();
        m_bMouseCaptured = FALSE;
    }
}

```

```

        if (m_nButtonState != BTN_UP)
            DrawButton(BTN_UP);
        ClientToScreen(&point);
        if (ScreenPointInButtonRect(point)) ButtonClicked();
    }
}

void CFileEditCtrl::OnMouseMove(UINT nFlags, CPoint point) {
    CEdit::OnMouseMove(nFlags, point);
    if (m_bMouseCaptured) {
        ClientToScreen(&point);
        if (ScreenPointInButtonRect(point)) {
            if (m_nButtonState != BTN_DOWN)
                DrawButton(BTN_DOWN);
        }
    }
    else if (m_nButtonState != BTN_UP) DrawButton(BTN_UP);
}

void CFileEditCtrl::OnNcCalcSize(BOOL bCalcValidRects, NCCALCSIZE_PARAMS FAR* lpncsp) {
    CEdit::OnNcCalcSize(bCalcValidRects, lpncsp);
    m_rcButtonRect = lpncsp->rgrc[0];
    if (m_bButtonLeft) {
        lpncsp->rgrc[0].left += (lpncsp->rgrc[0].bottom - lpncsp->rgrc[0].top) * 8/10;
        m_rcButtonRect.right = lpncsp->rgrc[0].left;
    }
    else {
        lpncsp->rgrc[0].right -= (lpncsp->rgrc[0].bottom - lpncsp->rgrc[0].top) * 8/10;
        m_rcButtonRect.left = lpncsp->rgrc[0].right;
    }
    //else
    if (bCalcValidRects)
        m_rcButtonRect.OffsetRect(-lpncsp->rgrc[1].left, -lpncsp->rgrc[1].top);
    m_rcButtonRect.NormalizeRect();
}

UINT CFileEditCtrl::OnNcHitTest(CPoint point) {
    UINT where = CEdit::OnNcHitTest(point);
    if (where == HTNOWHERE && ScreenPointInButtonRect(point))
        where = HTBORDER;
    return where;
}

void CFileEditCtrl::OnNcLButtonDown(UINT nHitTest, CPoint point) {
    CEdit::OnNcLButtonDown(nHitTest, point);
    if (ScreenPointInButtonRect(point)) {
        SetCapture();
        m_bMouseCaptured = TRUE;
        SetFocus();
        DrawButton(BTN_DOWN);
    }
}

void CFileEditCtrl::OnNcPaint() {
    CEdit::OnNcPaint();
    DrawButton(m_nButtonState);
}

void CFileEditCtrl::OnSetFocus(CWnd* pOldWnd) {
    CEdit::OnSetFocus(pOldWnd);
}

```



```

        SetSel(0,-1);
    }

    BOOL CFileEditCtrl::ScreenPointInButtonRect(CPoint point){
        CRect ControlRect;
        GetWindowRect(&ControlRect);
        point.Offset(-ControlRect.left, -ControlRect.top);
        return m_rcButtonRect.PtInRect(point);
    }

    BOOL CFileEditCtrl::SetFlags(DWORD dwFlags){
        if (m_bCreatingControl) {
            m_bCreatingControl = FALSE;
            m_bButtonLeft = dwFlags & FEC_BUTTONLEFT;
        }
        //if
        BOOL bFindFolder = dwFlags & FEC_FOLDER;
        if (bFindFolder){
            ASSERT (!(dwFlags & FEC_FILE));
            m_bTrailingSlash = dwFlags & FEC_TRAILINGSLASH;
            if (m_bFindFolder == bFindFolder && m_pBROWSEINFO) return TRUE;

            m_pBROWSEINFO = new BROWSEINFO();
            if (!m_pBROWSEINFO) return FALSE;
            if (m_pCFileDialog) {
                delete m_pCFileDialog;
                m_pCFileDialog = NULL;
            }
            //if
            ::ZeroMemory(m_pBROWSEINFO, sizeof(BROWSEINFO));
            m_pBROWSEINFO->hwndOwner = GetSafeHwnd();
            m_pBROWSEINFO->ulFlags = BIF_RETURNONLYFSDIRS;
        }
        //if
        else {
            ASSERT (dwFlags & FEC_FILE);
            if (m_bFindFolder == bFindFolder && m_pCFileDialog) return TRUE;
            CString szFilter;
            #if defined FEC_NORESOURCESTRINGS
                szFilter = FEC_IDS_ALLFILES;
            #else
                szFilter.LoadString(FEC_IDS_ALLFILES);
            #endif
            m_pCFileDialog = new CFECFileDialog(TRUE,
                NULL,
                NULL,
                OFN_HIDEREADONLY | OFN_FILEMUSTEXIST | OFN_NOCHANGEDIR,
                szFilter,
                this);
            if (!m_pCFileDialog) return FALSE;
            if (m_pBROWSEINFO){
                delete m_pBROWSEINFO;
                m_pBROWSEINFO = NULL;
            }
            //if
            m_pCFileDialog->m_ofn.hwndOwner = GetSafeHwnd();
            #if defined FEC_NORESOURCESTRINGS
                m_szCaption = FEC_IDS_FILEDIALOGTITLE;
            #else
                m_szCaption.LoadString(FEC_IDS_FILEDIALOGTITLE);
            #endif
            m_pCFileDialog->m_ofn.lpstrTitle = (LPCTSTR)m_szCaption;
        }
        //else
    }

```



```

        m_bFindFolder = bFindFolder;
        return TRUE;
    }

void CFileEditCtrl::SetWindowText(LPCTSTR lpszString) {
    m_bTextChanged = TRUE;
    CEdit::SetWindowText(lpszString);
}

void DDV_FileEditCtrl(CDataExchange *pDX, int nIDC) {
    CWnd *pWnd = pDX->m_pDlgWnd->GetDlgItem(nIDC);
    ASSERT (pWnd);
    if (!pWnd->IsKindOf(RUNTIME_CLASS(CFileEditCtrl))) {
        TRACE ("Control %d not subclassed to CFileEditCtrl, must first call\n");
        DDX_FileEditCtrl(pDX, nIDC);
        ASSERT(FALSE);
        AfxThrowNotSupportedException();
    }
    if (!pDX->m_bSaveAndValidate) return;
    CFileEditCtrl *pFEC = (CFileEditCtrl *)pWnd;
    pDX->PrepareEditCtrl(nIDC);
    POSITION pos = pFEC->GetStartPosition();
    if (!pos) {
        AfxMessageBox(FEC_IDS_NOFILE);
        pDX->Fail();
    }
    while(pos) {
        CString szMessage;
        CString szFile = pFEC->GetNextPathName(pos);
        DWORD dwAttribute = GetFileAttributes(szFile);
        if (dwAttribute == 0xFFFFFFFF) {
            szMessage.Format(FEC_IDS_NOTEXIST, szFile);
            AfxMessageBox(szMessage);
            pDX->Fail();
        }
        if ((pFEC->GetFlags() & FEC_FOLDER) && !(dwAttribute &
FILE_ATTRIBUTE_DIRECTORY)) {
            szMessage.Format(FEC_IDS_NOTFOLDER, szFile);
            AfxMessageBox(szMessage);
            pDX->Fail();
        }
        if ((pFEC->GetFlags() & FEC_FILE) && (dwAttribute &
FILE_ATTRIBUTE_DIRECTORY)) {
            szMessage.Format(FEC_IDS_NOTFILE, szFile);
            AfxMessageBox(szMessage);
            pDX->Fail();
        }
    }
}

void DDX_FileEditCtrl(CDataExchange *pDX, int nIDC, CString& rStr, DWORD dwFlags){
    CWnd *pWnd = pDX->m_pDlgWnd->GetDlgItem(nIDC);
    ASSERT (pWnd);
    if (pDX->m_bSaveAndValidate) {
        ASSERT(pWnd->IsKindOf(RUNTIME_CLASS(CFileEditCtrl)));
        rStr.Empty();
    }
}

```

```

        CFileEditCtrl *pFEC = (CFileEditCtrl *)pWnd;
        POSITION pos = pFEC->GetStartPosition();
        if (pos) rStr = pFEC->GetNextPathName(pos);
    }
    else {
        CFileEditCtrl *pFEC = NULL;
        if (!pWnd->IsKindOf(RUNTIME_CLASS(CFileEditCtrl))) {
            HWND hWnd = pDX->PrepareEditCtrl(nIDC);
            pFEC = new CFileEditCtrl(TRUE);
            if (!pFEC->SubclassWindow(hWnd)) {

                ASSERT(FALSE);
                AfxThrowNotSupportedException();
            }
            pFEC->SetFlags(dwFlags);
            pFEC->SetWindowPos(NULL,0,0,0,SWP_FRAMECHANGED|SWP_NOMOVE|SWP_NOSIZE|SWP_NOZORDER|SWP_NOACTIVATE);
        }
        else pFEC = (CFileEditCtrl *)pWnd;
        if (pFEC) pFEC->SetWindowText(rStr);
    }
}

void DDX_FileEditCtrl(CDataExchange *pDX, int nIDC, CFileEditCtrl &rCFEC, DWORD dwFlags){
    ASSERT (pDX->m_pDlgWnd->GetDlgItem(nIDC));
    if (rCFEC.m_hWnd == NULL) {
        ASSERT (!pDX->m_bSaveAndValidate);
        HWND hWnd = pDX->PrepareEditCtrl(nIDC);
        if (!rCFEC.SubclassWindow(hWnd)) {
            ASSERT(FALSE);
            AfxThrowNotSupportedException();
        }
    }
    rCFEC.SetFlags(dwFlags);

    rCFEC.SetWindowPos(NULL,0,0,0,SWP_FRAMECHANGED|SWP_NOMOVE|SWP_NOSIZE|SWP_NOZORDER|SWP_NOACTIVATE);
    if (pDX->m_bSaveAndValidate) rCFEC.GetStartPosition();
}

IMPLEMENT_DYNAMIC(CFECFileDialog, CFileDialog)

CFECFileDialog::CFECFileDialog(BOOL bOpenFileDialog, LPCTSTR lpszDefExt, LPCTSTR lpszFileName,
    DWORD dwFlags, LPCTSTR lpszFilter, CWnd* pParentWnd) :
    CFileDialog(bOpenFileDialog, lpszDefExt, lpszFileName, dwFlags, lpszFilter,
    pParentWnd) {
}

BEGIN_MESSAGE_MAP(CFECFileDialog, CFileDialog)
    //{{AFX_MSG_MAP(CFECFileDialog)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CFECFileDialog::OnInitDialog() {
    CFileDialog::OnInitDialog();
    if (!(m_ofn.Flags & OFN_EXPLORER)) {

```

```

        CString szOkButton;
#ifdef FEC_NORESOURCESTRINGS
        szOkButton = FEC_IDS_OKBUTTON;
#else
        szOkButton.LoadString(FEC_IDS_OKBUTTON);
#endif
        GetDlgItem(IDOK)->SetWindowText(szOkButton);
    }
    return TRUE;
}

void CFECFileDialog::OnInitDone(){
    CString szOkButton;
#ifdef FEC_NORESOURCESTRINGS
    szOkButton = FEC_IDS_OKBUTTON;
#else
    szOkButton.LoadString(FEC_IDS_OKBUTTON);
#endif
    CommDlg_OpenSave_SetControlText(GetParent()->m_hWnd, IDOK, (LPCTSTR)szOkButton);
}

```

Archivo de Encabezado FileEditCtrl.h

```
#if !defined(AFX_FILEEDITCTRL_H_F15965B0_B05A_11D4_B625_A1459D96AB20_INCLUDED_)
#define AFX_FILEEDITCTRL_H_F15965B0_B05A_11D4_B625_A1459D96AB20_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CFECFileDialog : public CFileDialog {
    DECLARE_DYNAMIC(CFECFileDialog)
public:
    CFECFileDialog(BOOL bOpenFileDialog,
        LPCTSTR lpszDefExt = NULL,
        LPCTSTR lpszFileName = NULL,
        DWORD dwFlags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
        LPCTSTR lpszFilter = NULL,
        CWnd* pParentWnd = NULL);
protected:
    ///{AFX_MSG(CFECFileDialog)
    virtual BOOL OnInitDialog();
    ///}AFX_MSG
    virtual void OnInitDone();
    DECLARE_MESSAGE_MAP()
};

// FEC_FILE - the control is used to find files, can not be used with
// FEC_FOLDER
// FEC_FOLDER - the control is used to find folders, can not be used with FEC_FILE
// FEC_TRAILINGSLASH - used with FEC_FOLDER, adds a slash to the end of the folder string
// FEC_BUTTONLEFT - places the button on the left side of the control, only used when control is
// being initialized
#define FEC_FILE 0x00000001
#define FEC_FOLDER 0x00000002
#define FEC_TRAILINGSLASH 0x00000004
#define FEC_BUTTONLEFT 0x00000008

class CFileEditCtrl : public CEdit {
    DECLARE_DYNAMIC(CFileEditCtrl)
// Construction
public:
    CFileEditCtrl(BOOL bAutoDelete = FALSE); // Constructor
// Attributes
public:
// Operations
public:
// Overrides
    // ClassWizard generated virtual function overrides
    ///{AFX_VIRTUAL(CFileEditCtrl)
    public:
    ///}AFX_VIRTUAL
    virtual BOOL Create(DWORD dwFlags, DWORD dwExStyle, LPCTSTR lpszWindowName,
        DWORD dwStyle, const RECT& rect, CWnd* pParentWnd, UINT nID);
// Implementation
public:
    void SetWindowText(LPCTSTR lpszString);
    virtual ~CFileEditCtrl();
    DWORD GetFlags();
    BROWSEINFO* GetBrowseInfo() const;
    CString GetNextPathName(POSITION &pos);
```



```

OPENFILENAME* GetOpenFileName() const;
POSITION GetStartPosition();
BOOL SetFlags(DWORD dwFlags);
protected:
    void ButtonClicked();
    void DrawButton (int nButtonState = 0);
    void DrawDots (CDC *pDC, COLORREF CR, int nOffset = 0);
    BOOL FECBrowseForFolder();
    BOOL FECOpenFile();
    void FillBuffers();
    BOOL ScreenPointInButtonRect(CPoint point);
    //{AFX_MSG(CFileEditCtrl)
    afx_msg BOOL OnChange();
    afx_msg void OnDestroy();
    afx_msg void OnDropFiles(HDROP hDropInfo);
    afx_msg void OnEnable(BOOL bEnable);
    afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnNcCalcSize(BOOL bCalcValidRects, NCCALCSIZE_PARAMS FAR* lpncsp);
    afx_msg UINT OnNcHitTest(CPoint point);
    afx_msg void OnNcLButtonDown(UINT nHitTest, CPoint point);
    afx_msg void OnNcPaint();
    afx_msg void OnSetFocus(CWnd* pOldWnd);
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    BOOL                m_bAutoDelete;
    BOOL                m_bButtonLeft;
    BOOL                m_bCreatingControl;
    BOOL                m_bFindFolder;
    BOOL                m_bMouseCaptured;
    BOOL                m_bTextChanged;
    BOOL                m_bTrailingSlash;
    LPTSTR              m_lpstrFiles;
    int                 m_nButtonState;
    BROWSEINFO*         m_pBROWSEINFO;
    CFECFileDialog* m_pCFileDialog;
    CRect               m_rcButtonRect;
    CString             m_szCaption;
    CString             m_szFolder;
};

void DDV_FileEditCtrl (CDataExchange *pDX, int nIDC);
void DDX_FileEditCtrl (CDataExchange *pDX, int nIDC, CFileEditCtrl &rCFEC, DWORD dwFlags);
void DDX_FileEditCtrl (CDataExchange *pDX, int nIDC, CString& rStr, DWORD dwFlags);

//{{AFX_INSERT_LOCATION}}
#endif //
!defined(AFX_FILEEDITCTRL_H__F15965B0_B05A_11D4_B625_A1459D96AB20__INCLUDED_)

```

Archivo FileEditDlg.cpp

Diálogo de Abrir un archivo

```
#include "stdafx.h"
#include "AgMp.h"
#include "FileEditDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CFileEditDlg::CFileEditDlg(CWnd* pParent /*=NULL*/) : CDialog(CFileEditDlg::IDD, pParent){
//{{AFX_DATA_INIT(CFileEditDlg)
m_file = _T("");
//}}AFX_DATA_INIT
}

void CFileEditDlg::DoDataExchange(CDataExchange* pDX){
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CFileEditDlg)
DDX_Control(pDX, IDOK, m_ok);
DDX_Control(pDX, IDCANCEL, m_cancel);
DDX_Text(pDX, IDC_file, m_file);
//}}AFX_DATA_MAP
DDX_FileEditCtrl(pDX, IDC_file, m_FileEditCtrl, FEC_FILE);
DDV_FileEditCtrl(pDX, IDC_file);
}

BEGIN_MESSAGE_MAP(CFileEditDlg, CDialog)
//{{AFX_MSG_MAP(CFileEditDlg)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CFileEditDlg::OnInitDialog() {
CDialog::OnInitDialog();
ASSERT((IDM_ABOUTBOX & 0xFFFF) == IDM_ABOUTBOX);
ASSERT(IDM_ABOUTBOX < 0xF000);
CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL) {
CString strAboutMenu;
strAboutMenu.LoadString(IDS_ABOUTBOX);
if (!strAboutMenu.IsEmpty()) {
pSysMenu->AppendMenu(MF_SEPARATOR);
pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
}
}

SetIcon(m_hIcon, TRUE);
SetIcon(m_hIcon, FALSE);
if (OPENFILENAME *ofn = m_FileEditCtrl.GetOpenFileName()) {
ofn->Flags |= OFN_ALLOWMULTISELECT;
buffer = new TCHAR[2000];
::ZeroMemory(buffer, 2000);
ofn->lpstrFile = buffer;
ofn->nMaxFile = 2000;
}
return TRUE;
}
```

```
}  
void CFileEditDlg::OnOK() {  
    UpdateData(TRUE);  
    DeleteFile("AgData.Dat");  
    int k=CopyFile(m_file,"AgData.Dat",1);  
    UpdateData(FALSE);  
    if (k==1) CDialog::OnOK();  
}
```

Archivo de Encabezado FileEditDlg.h

```
#include "RoundButton.h"
#include "FileEditCtrl.h"
#if
!defined(AFX_FILEEDITDLG_H__164DD401_546F_11D5_AB3A_C355CCD5B145__INCLUDED_)
#define AFX_FILEEDITDLG_H__164DD401_546F_11D5_AB3A_C355CCD5B145__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CFileEditDlg : public CDialog{
// Construction
public:
    CFileEditDlg(CWnd* pParent = NULL); // standard constructor
    CString m_String;
    CFileEditCtrl m_FileEditCtrl;
    TCHAR *buffer;
    virtual ~CFileEditDlg() { if (buffer) delete buffer; }

// Dialog Data
   //{{AFX_DATA(CFileEditDlg)
    enum { IDD = IDD_FILEEDIT_DIALOG };
    CRoundButton m_ok;
    CRoundButton m_cancel;
    CString m_file;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFileEditDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon;
    // Generated message map functions
   //{{AFX_MSG(CFileEditDlg)
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
!defined(AFX_FILEEDITDLG_H__164DD401_546F_11D5_AB3A_C355CCD5B145__INCLUDED_)
```


Archivo Roundbutton.cpp

```
#include "stdafx.h"
#include "math.h"
#include "RoundButton.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// prototypes
COLORREF GetColour(double dAngle, COLORREF crBright, COLORREF crDark);
void DrawCircle(CDC* pDC, CPoint p, LONG IRadius, COLORREF crColour, BOOL bDashed = FALSE);
void DrawCircleLeft(CDC* pDC, CPoint p, LONG IRadius, COLORREF crBright, COLORREF crDark);
void DrawCircleRight(CDC* pDC, CPoint p, LONG IRadius, COLORREF crBright, COLORREF crDark);

COLORREF GetColour(double dAngle, COLORREF crBright, COLORREF crDark){
#define Rad2Deg 180.0/3.1415

    ///define LIGHT_SOURCE_ANGLE      -2.356          // -2.356 radians = -135 degrees, i.e.
    From top left
    #define LIGHT_SOURCE_ANGLE -1.88

    ASSERT(dAngle > -3.1416 && dAngle < 3.1416);
    double dAngleDifference = LIGHT_SOURCE_ANGLE - dAngle;

    if (dAngleDifference < -3.1415) dAngleDifference = 6.293 + dAngleDifference;
    else if (dAngleDifference > 3.1415) dAngleDifference = 6.293 - dAngleDifference;

    double Weight = 0.5*(cos(dAngleDifference)+1.0);

    BYTE Red = (BYTE) (Weight*GetRValue(crBright) + (1.0-Weight)*GetRValue(crDark));
    BYTE Green = (BYTE) (Weight*GetGValue(crBright) + (1.0-Weight)*GetGValue(crDark));
    BYTE Blue = (BYTE) (Weight*GetBValue(crBright) + (1.0-Weight)*GetBValue(crDark));

    //TRACE("LightAngle = %0.0f, Angle = %3.0f, Diff = %3.0f, Weight = %0.2f, RGB
    %3d,%3d,%3d\n",
    //      LIGHT_SOURCE_ANGLE*Rad2Deg, dAngle*Rad2Deg,
    dAngleDifference*Rad2Deg, Weight, Red, Green, Blue);

    return RGB(Red, Green, Blue);
}

void DrawCircle(CDC* pDC, CPoint p, LONG IRadius, COLORREF crColour, BOOL bDashed){
    const int nDashLength = 1;
    LONG IError, IXoffset, IYoffset;
    int nDash = 0;
    BOOL bDashOn = TRUE;

    ASSERT( (p.x + IRadius <= LONG_MAX) && (p.y + IRadius <= LONG_MAX) );
    ASSERT( (p.x - IRadius >= LONG_MIN) && (p.y - IRadius >= LONG_MIN) );

    IXoffset = IRadius;
    IYoffset = 0;
    IError = -IRadius;
```

```

do {
    if (bDashOn) {
        pDC->SetPixelV(p.x + lXoffset, p.y + lYoffset, crColour);
        pDC->SetPixelV(p.x + lXoffset, p.y - lYoffset, crColour);
        pDC->SetPixelV(p.x + lYoffset, p.y + lXoffset, crColour);
        pDC->SetPixelV(p.x + lYoffset, p.y - lXoffset, crColour);
        pDC->SetPixelV(p.x - lYoffset, p.y + lXoffset, crColour);
        pDC->SetPixelV(p.x - lYoffset, p.y - lXoffset, crColour);
        pDC->SetPixelV(p.x - lXoffset, p.y + lYoffset, crColour);
        pDC->SetPixelV(p.x - lXoffset, p.y - lYoffset, crColour);
    }

    //Advance the error term and the constant X axis step
    lError += lYoffset++;

    //Check to see if error term has overflowed
    if ((lError += lYoffset) >= 0)
        lError -= -lXoffset * 2;

    if (bDashed && (++nDash == nDashLength)) {
        nDash = 0;
        bDashOn = !bDashOn;
    }
} while (lYoffset <= lXoffset);    //Continue until halfway point
}

void DrawCircleRight(CDC* pDC, CPoint p, LONG lRadius, COLORREF crBright, COLORREF crDark)
{
    LONG lError, lXoffset, lYoffset;

    //Check to see that the coordinates are valid
    ASSERT( (p.x + lRadius <= LONG_MAX) && (p.y + lRadius <= LONG_MAX) );
    ASSERT( (p.x - lRadius >= LONG_MIN) && (p.y - lRadius >= LONG_MIN) );

    //Set starting values
    lXoffset = lRadius;
    lYoffset = 0;
    lError = -lRadius;

    do {
        const double Pi = 3.141592654,
                    Pi_on_2 = Pi * 0.5,
                    Three_Pi_on_2 = Pi * 1.5;

        COLORREF crColour;
        double dAngle = atan2(lYoffset, lXoffset);

        //Draw the current pixel, reflected across all four arcs

        crColour = GetColour(dAngle, crBright, crDark);
        pDC->SetPixelV(p.x + lXoffset, p.y + lYoffset, crColour);

        crColour = GetColour(Pi_on_2 - dAngle, crBright, crDark);
        pDC->SetPixelV(p.x + lYoffset, p.y + lXoffset, crColour);

        crColour = GetColour(-Pi_on_2 + dAngle, crBright, crDark);
        pDC->SetPixelV(p.x + lYoffset, p.y - lXoffset, crColour);
    }

```

```

        crColour = GetColour(-dAngle, crBright, crDark);
        pDC->SetPixelV(p.x + IOffset, p.y - IYoffset, crColour);

        //Advance the error term and the constant X axis step
        IError += IYoffset++;

        //Check to see if error term has overflowed
        if ((IError += IYoffset) >= 0)
            IError -= -IXoffset * 2;

    } while (IYoffset <= IXoffset);    //Continue until halfway point
}

void DrawCircleLeft(CDC* pDC, CPoint p, LONG IRadius, COLORREF crBright, COLORREF crDark) {
    LONG IError, IOffset, IYoffset;

    ASSERT( (p.x + IRadius <= LONG_MAX) && (p.y + IRadius <= LONG_MAX) );
    ASSERT( (p.x - IRadius >= LONG_MIN) && (p.y - IRadius >= LONG_MIN) );

    IOffset = IRadius;
    IYoffset = 0;
    IError = -IRadius;

    do {
        const double Pi = 3.141592654,
                    Pi_on_2 = Pi * 0.5,
                    Three_Pi_on_2 = Pi * 1.5;
        COLORREF crColour;
        double dAngle = atan2(IYoffset, IOffset);

        //Draw the current pixel, reflected across all eight arcs

        crColour = GetColour(Pi_on_2 + dAngle, crBright, crDark);
        pDC->SetPixelV(p.x - IYoffset, p.y + IOffset, crColour);

        crColour = GetColour(Pi - dAngle, crBright, crDark);
        pDC->SetPixelV(p.x - IOffset, p.y + IYoffset, crColour);

        crColour = GetColour(-Pi + dAngle, crBright, crDark);
        pDC->SetPixelV(p.x - IOffset, p.y - IYoffset, crColour);

        crColour = GetColour(-Pi_on_2 - dAngle, crBright, crDark);
        pDC->SetPixelV(p.x - IYoffset, p.y - IOffset, crColour);

        //Advance the error term and the constant X axis step
        IError += IYoffset++;

        //Check to see if error term has overflowed
        if ((IError += IYoffset) >= 0)
            IError -= -IXoffset * 2;

    } while (IYoffset <= IXoffset);    //Continue until halfway point
}

CRoundButton::CRoundButton() {m_bDrawDashedFocusCircle = TRUE;}
CRoundButton::~CRoundButton() {m_rgn.DeleteObject();}

BEGIN_MESSAGE_MAP(CRoundButton, CButton)

```



```

    //{AFX_MSG_MAP(CRoundButton)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

void CRoundButton::PreSubclassWindow() {
    CButton::PreSubclassWindow();
    ModifyStyle(0, BS_OWNERDRAW);
    CRect rect;
    GetClientRect(rect);
    m_bStretch = rect.Width() > rect.Height() ? TRUE : FALSE;
    if(!m_bStretch) rect.bottom = rect.right = min(rect.bottom, rect.right);
    m_ptCentre = m_ptLeft = m_ptRight = rect.CenterPoint();
    m_nRadius = rect.bottom/2-1;
    m_ptLeft.x = m_nRadius;
    m_ptRight.x = rect.right - m_nRadius - 1;
    m_rgn.DeleteObject();
    SetWindowRgn(NULL, FALSE);
    m_rgn.CreateEllipticRgnIndirect(rect);
    SetWindowRgn(m_rgn, TRUE);
    ClientToScreen(rect);
    CWnd* pParent = GetParent();
    if (pParent) pParent->ScreenToClient(rect);
    if(!m_bStretch) MoveWindow(rect.left, rect.top, rect.Width(), rect.Height(), TRUE);
}

```

```

void CRoundButton::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct) {
    ASSERT(lpDrawItemStruct != NULL);

    CDC* pDC = CDC::FromHandle(lpDrawItemStruct->hDC);
    CRect rect = lpDrawItemStruct->rcItem;
    UINT state = lpDrawItemStruct->itemState;
    UINT nStyle = GetStyle();
    int nRadius = m_nRadius;

    int nSavedDC = pDC->SaveDC();

    pDC->SelectStockObject(NULL_BRUSH);
    pDC->FillSolidRect(rect, ::GetSysColor(COLOR_BTNFACE));

    // Draw the focus circle around the button for non-stretched buttons
    if ((state & ODS_FOCUS) && m_bDrawDashedFocusCircle && !m_bStretch)
        DrawCircle(pDC, m_ptCentre, nRadius, RGB(0,0,0));

    // Draw the raised/sunken edges of the button (unless flat)
    if (nStyle & BS_FLAT) {

        // for stretched buttons: draw left and right arcs and connect the with lines
        if(m_bStretch)
        {
            CPen* oldpen;

            CRect LeftBound(0,0,nRadius*2,nRadius*2);
            CRect RightBound(m_ptRight.x-nRadius,0,m_ptRight.x+nRadius,nRadius*2);

            oldpen = pDC->SelectObject(new CPen(PS_SOLID, 1,
            ::GetSysColor(COLOR_3DDKSHADOW)));
            pDC->Arc(LeftBound, CPoint(m_ptLeft.x,0), CPoint(m_ptLeft.x,nRadius*2));
            pDC->Arc(RightBound, CPoint(m_ptRight.x,nRadius*2),
            CPoint(m_ptRight.x,0));
        }
    }
}

```

```

        pDC->MoveTo(m_ptLeft.x,0); pDC->LineTo(m_ptRight.x,0);
        pDC->MoveTo(m_ptLeft.x,nRadius*2-1); pDC-
>LineTo(m_ptRight.x,nRadius*2-1);

        nRadius--;
        LeftBound.DeflateRect(1,1);
        RightBound.DeflateRect(1,1);

        delete pDC->SelectObject(new CPen(PS_SOLID, 1,
::GetSysColor(COLOR_3DHIGHLIGHT)));
        pDC->Arc(LeftBound, CPoint(m_ptLeft.x,1), CPoint(m_ptLeft.x,nRadius*2));
        pDC->Arc(RightBound, CPoint(m_ptRight.x,nRadius*2),
CPoint(m_ptRight.x,0));
        pDC->MoveTo(m_ptLeft.x,1); pDC->LineTo(m_ptRight.x,1);
        pDC->MoveTo(m_ptLeft.x,nRadius*2); pDC-
>LineTo(m_ptRight.x,nRadius*2);

        delete pDC->SelectObject(oldpen);
    }

    // for non-stretched buttons: draw two circles
    else
    {
        DrawCircle(pDC, m_ptCentre, nRadius--,
::GetSysColor(COLOR_3DDKSHADOW));
        DrawCircle(pDC, m_ptCentre, nRadius--,
::GetSysColor(COLOR_3DHIGHLIGHT));
    }
} else {
    if ((state & ODS_SELECTED)) {

        // draw the circular segments for stretched AND non-stretched buttons
        DrawCircleLeft(pDC, m_ptLeft, nRadius,
::GetSysColor(COLOR_3DDKSHADOW));
        DrawCircleRight(pDC, m_ptRight, nRadius,
::GetSysColor(COLOR_3DDKSHADOW));
        DrawCircleLeft(pDC, m_ptLeft, nRadius-1,
::GetSysColor(COLOR_3DSHADOW));
        DrawCircleRight(pDC, m_ptRight, nRadius-1,
::GetSysColor(COLOR_3DSHADOW));
        DrawCircleLeft(pDC, m_ptLeft, nRadius,
::GetSysColor(COLOR_3DLIGHT));
        DrawCircleRight(pDC, m_ptRight, nRadius,
::GetSysColor(COLOR_3DLIGHT));

        // draw connecting lines for stretched buttons only
        if (m_bStretch) {
            CPen* oldpen;
            //CPen* mypen;

            oldpen = pDC->SelectObject(new CPen(PS_SOLID, 1,
::GetSysColor(COLOR_3DDKSHADOW)));
            pDC->MoveTo(m_ptLeft.x, 1); pDC->LineTo(m_ptRight.x, 1);

            delete pDC->SelectObject(new CPen(PS_SOLID, 1,
::GetSysColor(COLOR_3DSHADOW)));
            pDC->MoveTo(m_ptLeft.x, 2);    pDC->LineTo(m_ptRight.x, 2);

```

```

        delete pDC->SelectObject(new CPen(PS_SOLID, 1,
::GetSysColor(COLOR_3DLIGHT)));
        pDC->MoveTo(m_ptLeft.x, m_ptLeft.y + nRadius-1); pDC-
>LineTo(m_ptRight.x, m_ptLeft.y + nRadius-1);

        delete pDC->SelectObject(new CPen(PS_SOLID, 1,
::GetSysColor(COLOR_3DHIGHLIGHT)));
        pDC->MoveTo(m_ptLeft.x, m_ptLeft.y + nRadius); pDC-
>LineTo(m_ptRight.x, m_ptLeft.y + nRadius);

        delete pDC->SelectObject(oldpen);
    }

    } else {

        // draw the circular segments for stretched AND non-stretched buttons
        DrawCircleLeft(pDC, m_ptLeft, nRadius,
            ::GetSysColor(COLOR_3DHIGHLIGHT),
::GetSysColor(COLOR_3DDKSHADOW));
        DrawCircleRight(pDC, m_ptRight, nRadius,
            ::GetSysColor(COLOR_3DHIGHLIGHT),
::GetSysColor(COLOR_3DDKSHADOW));
        DrawCircleLeft(pDC, m_ptLeft, nRadius - 1,
            ::GetSysColor(COLOR_3DLIGHT),
::GetSysColor(COLOR_3DSHADOW));
        DrawCircleRight(pDC, m_ptRight, nRadius - 1,
            ::GetSysColor(COLOR_3DLIGHT),
::GetSysColor(COLOR_3DSHADOW));

        // draw connecting lines for stretched buttons only
        if (m_bStretch)
        {
            CPen* oldpen;

            oldpen = pDC->SelectObject(new CPen(PS_SOLID, 1, pDC-
>GetPixel(m_ptLeft.x, 1)));
            pDC->MoveTo(m_ptLeft.x, 1); pDC->LineTo(m_ptRight.x, 1);

            delete pDC->SelectObject(new CPen(PS_SOLID, 1, pDC-
>GetPixel(m_ptLeft.x, 2)));
            pDC->MoveTo(m_ptLeft.x, 2); pDC->LineTo(m_ptRight.x, 2);

            delete pDC->SelectObject(new CPen(PS_SOLID, 1, pDC-
>GetPixel(m_ptLeft.x, m_ptLeft.y + nRadius)));
            pDC->MoveTo(m_ptLeft.x, m_ptLeft.y + nRadius); pDC-
>LineTo(m_ptRight.x, m_ptLeft.y + nRadius);

            delete pDC->SelectObject(new CPen(PS_SOLID, 1, pDC-
>GetPixel(m_ptLeft.x, m_ptLeft.y + nRadius - 1)));
            pDC->MoveTo(m_ptLeft.x, m_ptLeft.y + nRadius - 1); pDC-
>LineTo(m_ptRight.x, m_ptLeft.y + nRadius - 1);

            delete pDC->SelectObject(oldpen);
        }
    }

    }

    // draw the text if there is any
    CString strText;

```



```

GetWindowText(strText);

if (!strText.IsEmpty())
{
    CRgn rgn;

    if (m_bStretch){
        rgn.CreateRectRgn(m_ptLeft.x-nRadius/2, m_ptCentre.y-nRadius,
            m_ptRight.x+nRadius/2, m_ptCentre.y+nRadius);}

    else{
        rgn.CreateEllipticRgn(m_ptCentre.x-nRadius, m_ptCentre.y-nRadius,
            m_ptCentre.x+nRadius, m_ptCentre.y+nRadius);}

    pDC->SelectClipRgn(&rgn);

    CSize Extent = pDC->GetTextExtent(strText);
    CPoint pt = CPoint( m_ptCentre.x - Extent.cx/2, m_ptCentre.y - Extent.cy/2 );

    if (state & ODS_SELECTED) pt.Offset(1,1);

    pDC->SetBkMode(TRANSPARENT);

    if (state & ODS_DISABLED)
        pDC->DrawState(pt, Extent, strText, DSS_DISABLED, TRUE, 0,
(HBRUSH)NULL);
    else
    {
        // changed this code to give the text a 3d-look
        COLORREF oldcol = pDC-
>SetTextColor(::GetSysColor(COLOR_3DHIGHLIGHT));
        pDC->TextOut(pt.x, pt.y, strText);
        pDC->SetTextColor(::GetSysColor(COLOR_3DDKSHADOW));
        pDC->TextOut(pt.x-1, pt.y-1, strText);
        pDC->SetTextColor(oldcol);
    }

    pDC->SelectClipRgn(NULL);
    rgn.DeleteObject();
}

// Draw the focus circle on the inside of the button if it is non-stretched
if ((state & ODS_FOCUS) && m_bDrawDashedFocusCircle && !m_bStretch)
    DrawCircle(pDC, m_ptCentre, nRadius-2, RGB(0,0,0), TRUE);

pDC->RestoreDC(nSavedDC);
}

```

Archivo de encabezado Roundbutton.h

```
#if
!defined(AFX_ROUNDButton_H__5254170E_59CF_11D1_ABBA_00A0243D1382__INCLUDED_)
#define AFX_ROUNDButton_H__5254170E_59CF_11D1_ABBA_00A0243D1382__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CRoundButton : public CButton{
// Construction
public:
    CRoundButton();
// Attributes
public:
// Operations
public:
// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CRoundButton)
    public:
        virtual void DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct);
    protected:
        virtual void PreSubclassWindow();
    //}AFX_VIRTUAL
// Implementation
public:
    virtual ~CRoundButton();
    CRgn m_rgn;
    CPoint m_ptCentre;
    CPoint m_ptLeft;
    CPoint m_ptRight;
    int m_nRadius;
    BOOL m_bDrawDashedFocusCircle;
    BOOL m_bStretch;
    // Generated message map functions
protected:
    //{AFX_MSG(CRoundButton)
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
!defined(AFX_ROUNDButton_H__5254170E_59CF_11D1_ABBA_00A0243D1382__INCLUDED_)
```


Archivo Crosavan.cpp

Configuración avanzada del crossovet

```
#include "stdafx.h"
#include "AgMp.h"
#include "Crossavan.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
int cros=0;

CCrossavan::CCrossavan(CWnd* pParent /*=NULL*/) : CDialog(CCrossavan::IDD, pParent){
//{{AFX_DATA_INIT(CCrossavan)
m_extremos = FALSE;
//}}AFX_DATA_INIT
}

void CCrossavan::DoDataExchange(CDataExchange* pDX){
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CCrossavan)
DDX_Control(pDX, IDC_1punto, m_ok);
DDX_Control(pDX, IDCANCEL, m_cancel);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCrossavan, CDialog)
//{{AFX_MSG_MAP(CCrossavan)
ON_BN_CLICKED(IDC_1punto, On1punto)
ON_BN_CLICKED(IDC_uniforme, Onuniforme)
ON_WM_SHOWWINDOW()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

void CCrossavan::Onuniforme() {cros=1;}
void CCrossavan::On1punto() {cros=0;}

void CCrossavan::OnShowWindow(BOOL bShow, UINT nStatus) {
FILE *qtr;
CDialog::OnShowWindow(bShow, nStatus);
CheckRadioButton(IDC_1punto, IDC_uniforme, IDC_1punto);
qtr=fopen("cro.agc", "rt");
if (qtr!=NULL) {
UpdateData(TRUE);
fscanf(qtr, "%d", &cros);
fscanf(qtr, "%d", &m_extremos);
fclose(qtr);
if (cros==1) CheckRadioButton(IDC_1punto, IDC_uniforme, IDC_uniforme);
if (m_extremos==TRUE) CheckDlgButton(IDC_extremos, TRUE);
UpdateData(FALSE);
}
}

void CCrossavan::OnOK() {
FILE *qtr;
UpdateData(TRUE);
```

```
UpdateData(FALSE);  
DeleteFile("cro.agc");  
qtr=fopen("cro.agc","a");  
fprintf(qtr,"%d\n",cros);  
fprintf(qtr,"%d",m_extremos);  
fclose(qtr);  
CDialog::OnOK();  
}
```

Archivo de Encabezado Crosavan.h

```
#include "RoundButton.h"
#if !defined(AFX_CROSSAVAN_H__22E576E2_1096_11D5_AB39_E98752277B54__INCLUDED_)
#define AFX_CROSSAVAN_H__22E576E2_1096_11D5_AB39_E98752277B54__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CCrossavan : public CDialog{
// Construction
public:
    CCrossavan(CWnd* pParent = NULL); // standard constructor
// Dialog Data
   //{{AFX_DATA(CCrossavan)
    enum { IDD = IDD_CROSSAVAN_DIALOG };
    CRoundButton    m_ok;
    CRoundButton    m_cancel;
    BOOL    m_extremos;
    //}}AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CCrossavan)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
protected:
    // Generated message map functions
   //{{AFX_MSG(CCrossavan)
    afx_msg void On1punto();
    afx_msg void Onuniforme();
    virtual void OnOK();
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
#endif(AFX_CROSSAVAN_H__22E576E2_1096_11D5_AB39_E98752277B54__INCLUDED_)
```

Archivo Mutavan.h

Configuración avanzada de la mutación

```
#include "stdafx.h"
#include "AgMp.h"
#include "Mutavan.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

int muta=0;

CMutavan::CMutavan(CWnd* pParent /*=NULL*/) : CDialog(CMutavan::IDD, pParent) {
    ///{{AFX_DATA_INIT(CMutavan)
    m_menor = FALSE;
    ///}}AFX_DATA_INIT
}

void CMutavan::DoDataExchange(CDataExchange* pDX) {
    CDialog::DoDataExchange(pDX);
    ///{{AFX_DATA_MAP(CMutavan)
    DDX_Control(pDX, IDOK, m_ok);
    DDX_Control(pDX, IDCANCEL, m_cancel);
    DDX_Check(pDX, IDC_menor, m_menor);
    ///}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMutavan, CDialog)
    ///{{AFX_MSG_MAP(CMutavan)
    ON_BN_CLICKED(IDC_1punto, On1punto)
    ON_BN_CLICKED(IDC_uniforme, Onuniforme)
    ON_WM_SHOWWINDOW()
    ///}}AFX_MSG_MAP
END_MESSAGE_MAP()

void CMutavan::On1punto() {muta=0;}
void CMutavan::Onuniforme() {muta=1;}

void CMutavan::OnShowWindow(BOOL bShow, UINT nStatus) {
    FILE *qtr;
    CDialog::OnShowWindow(bShow, nStatus);
    CheckRadioButton(IDC_1punto, IDC_uniforme, IDC_1punto);
    qtr=fopen("mut.agc", "rt");
    if (qtr!=NULL) {
        UpdateData(TRUE);
        fscanf(qtr, "%d", &muta);
        fscanf(qtr, "%d", &m_menor);
        fclose(qtr);
        if (muta==1) CheckRadioButton(IDC_1punto, IDC_uniforme, IDC_uniforme);
        if (m_menor==TRUE) CheckDlgButton(IDC_menor, TRUE);
        UpdateData(FALSE);
    }
    ///f
}
```

```
void CMutavan::OnOK() {  
FILE *qtr;  
    UpdateData(TRUE);  
    UpdateData(FALSE);  
    DeleteFile("mut.Agc");  
    qtr=fopen("mut.Agc","at");  
    fprintf(qtr,"%d\n",muta);  
    fprintf(qtr,"%d",m_menor);  
    fclose(qtr);  
    CDialog::OnOK();  
}
```

Archivo de encabezado Mutavan.h

```
#include "RoundButton.h"
#if !defined(AFX_MUTAVAN_H_22E576E1_1096_11D5_AB39_E98752277B54_INCLUDED_)
#define AFX_MUTAVAN_H_22E576E1_1096_11D5_AB39_E98752277B54_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMutavan : public CDialog{
// Construction
public:
    CMutavan(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    ///{{AFX_DATA(CMutavan)
    enum { IDD = IDD_MUTAVAN_DIALOG };
    CRoundButton m_ok;
    CRoundButton m_cancel;
    BOOL m_menor;
    ///}}AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
    ///{{AFX_VIRTUAL(CMutavan)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    ///}}AFX_VIRTUAL
// Implementation
protected:
    // Generated message map functions
    ///{{AFX_MSG(CMutavan)
    afx_msg void On1punto();
    afx_msg void Onuniforme();
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    virtual void OnOK();
    ///}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

///{{AFX_INSERT_LOCATION}}
#endif //
!defined(AFX_MUTAVAN_H_22E576E1_1096_11D5_AB39_E98752277B54_INCLUDED_)
```


Archivo Simulador.cpp

Simulador Poblacional

```
#include "stdafx.h"
#include "AgMp.h"
#include "Simulador.h"
#include "Mutavan.h"
#include "Crossavan.h"
#include "Resulta.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

int selec;

CSimulador::CSimulador(CWnd* pParent /*=NULL*/): CDialog(CSimulador::IDD, pParent){
   //{{AFX_DATA_INIT(CSimulador)
    m_iter = 25;
    m_pcross = 0.8;
    m_pmuta = 0.1;
    m_elitismo = FALSE;
    m_nhombres = 250;
    m_nmujeres = 230;
    m_nhijos = 3.3;
    m_nacea = 14.0;
    m_naceb = 44.0;
    m_muere = 80.0;
    m_best = FALSE;
    m_maximo = 0.0;
   //}}AFX_DATA_INIT
}

void CSimulador::DoDataExchange(CDataExchange* pDX){
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CSimulador)
    DDX_Control(pDX, IDC_Mutavan, m_mutavan);
    DDX_Control(pDX, IDC_Crossavan, m_crossavan);
    DDX_Control(pDX, IDOK, m_ejecutar);
    DDX_Control(pDX, IDCANCEL, m_salir);
    DDX_Text(pDX, IDC_Generaciones, m_iter);
    DDX_Text(pDX, IDC_PCros, m_pcross);
    DDV_MinMaxDouble(pDX, m_pcross, 0., 1.);
    DDX_Text(pDX, IDC_Pmut, m_pmuta);
    DDV_MinMaxDouble(pDX, m_pmuta, 0., 1.);
    DDX_Check(pDX, IDC_elitismo, m_elitismo);
    DDX_Text(pDX, IDC_Nhombres, m_nhombres);
    DDX_Text(pDX, IDC_NMujeres, m_nmujeres);
    DDX_Text(pDX, IDC_Nhijos, m_nhijos);
    DDX_Text(pDX, IDC_Nacea, m_nacea);
    DDX_Text(pDX, IDC_Naceb, m_naceb);
    DDX_Text(pDX, IDC_Muere, m_muere);
    DDX_Check(pDX, IDC_best, m_best);
    DDX_Text(pDX, IDC_Maximo, m_maximo);
   //}}AFX_DATA_MAP
}
```

```

BEGIN_MESSAGE_MAP(CSimulador, CDialog)
   //{{AFX_MSG_MAP(CSimulador)
    ON_WM_SHOWWINDOW()
    ON_BN_CLICKED(IDC_Rule, OnRule)
    ON_BN_CLICKED(IDC_Torne, OnTorne)
    ON_BN_CLICKED(IDC_Crossavan, OnCrossavan)
    ON_BN_CLICKED(IDC_Mutavan, OnMutavan)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_EVENTSINK_MAP(CSimulador, CDialog)
   //{{AFX_EVENTSINK_MAP(CSimulador)
   //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()

void CSimulador::OnRule() {selec=0;}
void CSimulador::OnTorne() {selec=1;}

void CSimulador::OnCrossavan() {
    CCrossavan Co;
    Co.DoModal();
}

void CSimulador::OnMutavan() {
    CMutavan Mu;
    Mu.DoModal();
}

void CSimulador::OnShowWindow(BOOL bShow, UINT nStatus) {
    CDialog::OnShowWindow(bShow, nStatus);
    selec=1;
    CheckRadioButton(IDC_Torne, IDC_Rule, IDC_Torne);
}

void CSimulador::OnOK() {
    FILE *qtr;
    int tcross=0, tmuta=0, extremos=0, peor=0;
    CResulta Re;
    UpdateData(TRUE);
    UpdateData(FALSE);
    qtr=fopen("Cro.Agc", "rt");
    if (qtr!=NULL) {
        fscanf(qtr, "%d", &tcross);
        fscanf(qtr, "%d", &extremos);
        fclose(qtr);
    }
    qtr=fopen("Mut.Agc", "rt");
    if (qtr!=NULL) {
        fscanf(qtr, "%d", &tmuta);
        fscanf(qtr, "%d", &peor);
        fclose(qtr);
    }
    DeleteFile("confi.Agc");
    qtr=fopen("confi.Agc", "at");
    fprintf(qtr, "%ld\n", m_iter);
    fprintf(qtr, "%d\n", tcross);
    fprintf(qtr, "%lf\n", m_pcross);
    fprintf(qtr, "%d\n", extremos);
}

```



```
fprintf(qtr,"%d\n",tmuta);
fprintf(qtr,"%lf\n",m_pmuta);
fprintf(qtr,"%d\n",peor);
fprintf(qtr,"%d\n",selec);
fprintf(qtr,"%d\n",m_best);
fprintf(qtr,"%d\n",m_elitismo);
fprintf(qtr,"%ld\n",m_nhombres);
fprintf(qtr,"%ld\n",m_nmujeres);
fprintf(qtr,"%lf\n",m_nhijos);
fprintf(qtr,"%lf\n",m_nacea);
fprintf(qtr,"%lf\n",m_naceb);
fprintf(qtr,"%lf\n",m_muere);
fprintf(qtr,"%lf",m_maximo);
fclose(qtr);
if (Re.Proceso()) Re.DoModal();
}
```

Archivo de encabezado Simulador.h

```
#include "RoundButton.h"
#ifndef AFX_SIMULADOR_H__2FD24BA1_2B65_11D5_AB39_FE351F0F8C62__INCLUDED_
#define AFX_SIMULADOR_H__2FD24BA1_2B65_11D5_AB39_FE351F0F8C62__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CSimulador : public CDialog {
// Construction
public:
    CSimulador(CWnd* pParent = NULL); // standard constructor
// Dialog Data
   //{{AFX_DATA(CSimulador)
    enum { IDD = IDD_SIMULADOR_DIALOG };
    CRoundButton m_mutavan;
    CRoundButton m_crossavan;
    CRoundButton m_ejecutar;
    CRoundButton m_salir;
    long m_iter;
    double m_pcross;
    double m_pmuta;
    BOOL m_elitismo;
    long m_nhombres;
    long m_nmujeres;
    double m_nhijos;
    double m_nacea;
    double m_naceb;
    double m_muere;
    BOOL m_best;
    double m_maximo;
    }{{AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CSimulador)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }{{AFX_VIRTUAL
// Implementation
protected:
    // Generated message map functions
   //{{AFX_MSG(CSimulador)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    virtual void OnOK();
    afx_msg void OnRule();
    afx_msg void OnTorne();
    afx_msg void OnCrossavan();
    afx_msg void OnMutavan();
    DECLARE_EVENTSINK_MAP()
    }{{AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
#ifdef AFX_SIMULADOR_H__2FD24BA1_2B65_11D5_AB39_FE351F0F8C62__INCLUDED_
```

Archivo de encabezado AgPob.hpp

Algoritmo genético de la Simulación

```
#define longicromo 4
```

```
class AGpob {
private:
    int npob;           //tamaño de la población
    int genera;         //# de generaciones
    double Pcro;        //probabilidad de crossover
    double Pmut;        //probabilidad de mutar
    int sele;           //tipo de selección
    int promejor;       //propagar el mejor?
    int limites;        //crossover solo en los limites?
    int Elitismo;       //Usar elitismo?
    int tipocros;       //tipo de crossover
    int tipomuta;       //tipo de mutacion
    int menor;          //mutar sólo los menores
    int exigir;         //exigir mutacion
    long iteracion;     //contador de las iteraciones
    double mayaptitud;  //mayor aptitud
    double proaptitud;  //aptitud promedio
    double proedad;     //edad promedio
    double proedadm;
    double suma;        //Suma de las aptitudes
    double nhombres,nmujeres;//numero de hombres y mujeres
    double nanterior;//numero anterior de la poblacion
    double hanterior;//numero anterior de hombres
    double manterior;//numero anterior de mujeres
    double logi_apti;//poblacion para aptitud minima
    double nacea,naceb;//edades en las que se reproducen
    double muere;//edad en la que muere
    double nhijos;//promedio de hijos
    double crecimiento;
    double crecih;
    double crecim;
    long nacen;
    long mueren;
    struct genetico {           //estructura de caracter genetico
        char cromosoma[longicromo+1]; //individuo
        char cromosoma2[longicromo+1]; //individuo que va a reproducirse
        double geno;            //genotipo
        double aptitud;         //aptitud
        double dispo;           //aptitud relativa
    };
    struct datos_persona {
        int edad_actual; //edad
        int edad_limite; //edad de fallecimiento
        char sexo;        //sexo
        int reproa;        //Intervalo en que puede reproducirse
        int reprob;
        int num_hijos;
        int num_tot_hijos;
    };
    struct persona {           //estructura de persona con cualidades
        genetico gene;         //parte reproductiva
        datos_persona datos; //parte de los datos
    };
};
```

```

};
struct poblacion {          //estructura de poblacion con personas
    persona info;           //cada persona
    poblacion *sig;
}*Hpob,*Mpob;              //Separo hombres y mujeres
struct selec {              //estructura de seleccion para Ruleta
    double acu;             //acumulada de aptitudes
    int pos;                //posicion
    selec *sig;
};

public:
    AGpob(void);
    double normal(double,double); //genera numeros normales
    //Operaciones de los Algoritmos Genéticos
    void lee_configuracion(void); //lee la configuración dada
    void lee_parametros(void);    //lee los parametros dados
    void generainicial(void);     //genera la poblacion inicial
    double aptitud(double x) {return sqrt(x)/10;}
    long random(int x) {return (long)floor(x*U01());}
    int muerte(double w) {return (int)(0.5*sqrt((double)(1+8*random((long)(w*(w+1)*0.5))))-0.5);}
    double U01(void){return (double)rand()/RAND_MAX;} //
uniforme U(0,1)
    int converge(void);          //verifica si se termina el AG
    int puede_reproducirse(poblacion *); //verifica si puede reproducirse
    long num_reproduce(int); //halla el número de individuos que pueden reproducirse
    void halla_aptitud01(void); //halla la aptitudes entre 0 y 1
    void seleccion_Torneo(long,long); //selecciona por medio del Torneo
    void seleccion_Ruleta(int,long); //selecciona por medio de la Ruleta
    void crossover1(void);        //Crossover en un punto
    void crossoverU(void);        //Crossover Uniforme
    void mutacion1(int);          //Mutacion en un punto
    void mutacionU(int);          //Mutacion Uniforme
    void pasa_tiempo(void);       //Función que envejece las personas
    void elimina_viejos(void);    //Elimina los que pasan el limite de edad
    double genotipo(char []);     //genotipo del cromosoma
    void reemplazo(void);         //reemplazo la antigua poblacion
    void halla_mayproaptitud(void); //halla la aptitud promedio y la mayor
    void guardar_resultados(int); //guarda los resultados de una iteracion
    //Operaciones tradicionales de listas enlazadas
    void crear(void);             //crea una poblacion
    void destruir(void);          //Destruye una poblacion
    void quitar_primeroh(void); //elimina el primero de la poblacion
    void quitar_primerom(void); //elimina el primero de la poblacion
    void agregar_hombre(persona); //agrega una persona
    void agregar_mujer(persona); //agrega una persona
    void ordenar(int,int); //ordena la poblacion
    //Operadores matematicos lógicos
    char conjuncion(char,char); //operador y
    char disyuncion(char,char); //operador o
    char no(char);               //negacion
    char igual(char,char);       //operador de igualdad
    char diferente(char,char); //operador de diferencia
    int _sele(void) {return sele;}
    int _tipomuta(void) {return tipomuta;}
    int _tipocros(void) {return tipocros;}
    int _genera(void) {return genera;}
    int Logistica_permite(void) {return (nhombres+nmujeres<logi_apti||logi_apti==0);}
};

```



```

AGpob::AGpob(void) {
    iteracion=0;
    nanterior=-1;
    hanterior=-1;
    manterior=-1;
}

void AGpob::lee_configuracion(void) {
    FILE *qtr;
    qtr=fopen("Confi.Agc","rt");
    rewind(qtr);
    fscanf(qtr,"%ld",&genera);
    fscanf(qtr,"%d",&tipocros);
    fscanf(qtr,"%lf",&Pcro);
    fscanf(qtr,"%d",&limites);
    fscanf(qtr,"%d",&tipomuta);
    fscanf(qtr,"%lf",&Pmut);
    fscanf(qtr,"%d",&menor);
    fscanf(qtr,"%d",&sele);
    fscanf(qtr,"%d",&promenor);
    fscanf(qtr,"%d",&elitismo);
    fscanf(qtr,"%lf",&nhombres);
    fscanf(qtr,"%lf",&nmujeres);
    fscanf(qtr,"%lf",&nhijos);
    fscanf(qtr,"%lf",&nacea);
    fscanf(qtr,"%lf",&naceb);
    fscanf(qtr,"%lf",&muere);
    fscanf(qtr,"%lf",&logi_apti);
    fclose(qtr);
}

void AGpob::generainicial(void) {
    int i,j,k;
    persona x;
    lee_configuracion();
    crear();
    iteracion=0;
    nanterior=-1;
    hanterior=-1;
    manterior=-1;
    for (i=0;i<nhombres;i++) {
        for (j=0;j<longicromo;j++) {
            k=random(2);
            if (k==0) x.gene.cromosoma[j]='0';
            else x.gene.cromosoma[j]='1';
        }
        x.gene.cromosoma[j]=NULL;
        x.gene.cromosoma2[0]=NULL;
        x.datos.edad_actual=(int)muere-abs((int)muerte(muere));
        do {
            x.datos.edad_limite=abs((int)muerte(muere));
        } while (x.datos.edad_limite<x.datos.edad_actual);
        do {
            x.datos.num_tot_hijos=(int)normal(nhijos,1);
        } while (x.datos.num_tot_hijos<0);
        if (x.datos.edad_actual>=(nacea+naceb)*0.5)
            x.datos.num_hijos=(int)(x.datos.num_tot_hijos);
        else if (x.datos.edad_actual>=nacea)
            x.datos.num_hijos=(int)(x.datos.num_tot_hijos*1.0/2.0);
    }
}

```

```

        else x.datos.num_hijos=0;
        do {
            x.datos.reproa=(int)nacea+random(10);
            x.datos.reprob=(int)naceb-random(10);
        } while (x.datos.reproa>=x.datos.reprob);
        x.datos.sexo='H';
        agregar_hombre(x);
    } //for
    for (i=0; i<nmujeres; i++) {
        for (j=0; j<longicromo; j++) {
            k=random(2);
            if (k==0) x.gene.cromosoma[j]='0';
            else x.gene.cromosoma[j]='1';
        } //for
        x.gene.cromosoma[j]=NULL;
        x.gene.cromosoma2[0]=NULL;
        x.datos.edad_actual=(int)muere-abs((int)muerte(muere));
        do {
            x.datos.edad_limite=abs((int)muerte(muere));
        } while (x.datos.edad_limite<x.datos.edad_actual);
        do {
            x.datos.num_tot_hijos=abs((int)normal(nhijos,1));
        } while (x.datos.num_tot_hijos<0);
        if (x.datos.edad_actual>=(nacea+naceb)*0.5)
            x.datos.num_hijos=(int)(x.datos.num_tot_hijos);
        else if (x.datos.edad_actual>=nacea)
            x.datos.num_hijos=(int)(x.datos.num_tot_hijos*1.0/2.0);
        else x.datos.num_hijos=0;
        do {
            x.datos.reproa=(int)nacea+random(10);
            x.datos.reprob=(int)naceb-random(10);
        } while (x.datos.reproa>=x.datos.reprob);
        x.datos.sexo='M';
        agregar_mujer(x);
    } //for
    nacen=0;
    mueren=0;
}

int AGpob::converge(void) {
    iteracion++;
    return (genera==iteracion);
}

void AGpob::halla_aptitud01(void) {
    poblacion *aux;
    aux=Hpob;
    suma=0.0;
    while (aux!=NULL) {
        suma+=aux->info.gene.aptitud;
        aux=aux->sig;
    } //while
    aux=Mpob;
    while (aux!=NULL) {
        suma+=aux->info.gene.aptitud;
        aux=aux->sig;
    } //while
    aux=Hpob;
    while (aux!=NULL) {

```

```

        aux->info.gene.dispo=aux->info.gene.aptitud/suma;
        aux=aux->sig;
    }//while
    aux=Mpob;
    while (aux!=NULL) {
        aux->info.gene.dispo=aux->info.gene.aptitud/suma;
        aux=aux->sig;
    }//while
}

void AGpob::seleccion_Torneo(long nhr,long nmr) {
    poblacion *aux;
    aux=Hpob;
    for (int k=0;k<nhr;k++) {
        if (aux->info.datos.edad_actual>=aux->info.datos.reproa&&aux-
        >info.datos.edad_actual<=aux->info.datos.reprob&&aux->info.datos.num_hijos<aux-
        >info.datos.num_tot_hijos) {
            aux->info.datos.num_hijos++;
            strcpy(aux->info.gene.cromosoma2,aux->info.gene.cromosoma);
        }//if
        else aux->info.gene.cromosoma2[0]=NULL;
        aux=aux->sig;
    }//for
    aux=Mpob;
    for (k=0;k<nmr;k++) {
        if (aux->info.datos.edad_actual>=aux->info.datos.reproa&&aux-
        >info.datos.edad_actual<=aux->info.datos.reprob&&aux->info.datos.num_hijos<aux-
        >info.datos.num_tot_hijos) {
            aux->info.datos.num_hijos++;
            strcpy(aux->info.gene.cromosoma2,aux->info.gene.cromosoma);
        }//if
        else aux->info.gene.cromosoma2[0]=NULL;
        aux=aux->sig;
    }//for
}

```

```

void AGpob::seleccion_Ruleta(int x,long nr) {
    selec *S,*S2;
    poblacion *aux,*aux2;
    double ant=0.0,U;
    int contador=0,i;
    long k;
    if (x==0) aux=Mpob;
    else aux=Hpob;
    S=(selec *)malloc(sizeof(selec));
    S->sig=NULL;
    S2=S;
    for (k=0;k<nr;k++) {
        S2->sig=(selec *)malloc(sizeof(selec));
        S2=S2->sig;
        S2->acu=aux->info.gene.aptitud+ant;
        ant=S2->acu;
        aux=aux->sig;
    }//for
    S2->sig=NULL;
    S2=S;
    for (k=0;k<nr;k++) S2->acu=(double)S2->acu/ant;
    if (x==0) aux=Mpob;
    else aux=Hpob;
}

```

```

while (aux!=NULL) {
    U=U01();
    S2=S;
    contador=0;
    while (S2->sig!=NULL) {
        S2=S2->sig;
        contador++;
        if (U<S2->acu) break;
    }//while
    if (x==0) aux2=Mpob;
    else aux2=Hpob;
    for (i=1;i<contador;i++) aux2=aux2->sig;
    if (aux2->info.datos.edad_actual>=aux2->info.datos.reproa&&aux2-
>info.datos.edad_actual<=aux2->info.datos.reprob&&aux2->info.datos.num_hijos<aux2-
>info.datos.num_tot_hijos) {
        aux2->info.datos.num_hijos++;
        strcpy(aux->info.gene.cromosoma2,aux2->info.gene.cromosoma);
    }//if
    else aux->info.gene.cromosoma2[0]=NULL;
    aux=aux->sig;
}

char AGpob::conjuncion(char a,char b) {
    if (a=='0' || b=='0') return '0';
    return '1';
}

char AGpob::disyuncion(char a,char b) {
    if (a=='1' || b=='1') return '1';
    return '0';
}

char AGpob::no(char b) {
    if (b=='0') return '1';
    return '0';
}

char AGpob::igual(char a,char b) {
    if (a==b) return '1';
    return '0';
}

char AGpob::diferente(char a,char b) {
    if (a!=b) return '1';
    return '0';
}

void AGpob::crossover1(void) {
    poblacion *auxH,*auxM;
    persona hijo;
    double U;
    char hijos[14][longicromo+1]; //posibles hijos matematicos
    int i,pos,k1,k2,k;
    U=U01();
    if (U<=Pcro) {
        if (limites==1) pos=longicromo-1;
        else pos=1+random(longicromo-1);
        auxH=Hpob;
    }
}

```



```

    auxM=Mpob;
    while (auxH!=NULL&&auxM!=NULL&&auxH-
>info.gene.cromosoma2[0]!=NULL&&auxM->info.gene.cromosoma2[0]!=NULL) {
        for (i=0;i<longicromo;i++) {
            hijos[0][i]=conjuncion(auxH-
>info.gene.cromosoma2[i],auxM->info.gene.cromosoma2[i]);
            hijos[1][i]=disyuncion(auxH-
>info.gene.cromosoma2[i],auxM->info.gene.cromosoma2[i]);
            hijos[2][i]=conjuncion(no(auxH-
>info.gene.cromosoma2[i]),no(auxM->info.gene.cromosoma2[i]));
            hijos[3][i]=disyuncion(no(auxH-
>info.gene.cromosoma2[i]),no(auxM->info.gene.cromosoma2[i]));
            hijos[4][i]=conjuncion(auxH-
>info.gene.cromosoma2[i],no(auxM->info.gene.cromosoma2[i]));
            hijos[5][i]=disyuncion(auxH-
>info.gene.cromosoma2[i],no(auxM->info.gene.cromosoma2[i]));
            hijos[6][i]=conjuncion(no(auxH-
>info.gene.cromosoma2[i]),auxM->info.gene.cromosoma2[i]);
            hijos[7][i]=disyuncion(no(auxH-
>info.gene.cromosoma2[i]),auxM->info.gene.cromosoma2[i]);
            hijos[8][i]= igual (auxH->info.gene.cromosoma2[i],auxM-
>info.gene.cromosoma2[i]);
            hijos[9][i]= diferente(auxH-
>info.gene.cromosoma2[i],auxM->info.gene.cromosoma2[i]);
            hijos[10][i]=auxH->info.gene.cromosoma2[i];
            hijos[11][i]=auxM->info.gene.cromosoma2[i];
            hijos[12][i]=no(auxH->info.gene.cromosoma2[i]);
            hijos[13][i]=no(auxM->info.gene.cromosoma2[i]);
        }
        for (i=0;i<14;i++) hijos[i][longicromo]=NULL;
        if (Elitismo==0||auxH->info.gene.apptitud<mayaptitud) {
            k1=(int)random(14);
            k2=(int)random(14);
        }
        else {k1=10;k2=11;}
        for (i=0;i<pos;i++) hijo.gene.cromosoma[i]=hijos[k1][i];
        for (i=pos;i<longicromo;i++) hijo.gene.cromosoma[i]=hijos[k2][i];
        hijo.gene.cromosoma[longicromo]=NULL;
        hijo.gene.cromosoma2[0]=NULL;
        hijo.datos.edad_actual=0;
        hijo.datos.edad_limite=muerte(muere);
        do {
            hijo.datos.num_tot_hijos=(int)normal(nhijos,1);
        } while(hijo.datos.num_tot_hijos<0);
        hijo.datos.num_hijos=0;
        do {
            hijo.datos.reproa=(int)nacea+random(10);
            hijo.datos.reprob=(int)naceb-random(10);
        } while (hijo.datos.reproa>=hijo.datos.reprob);
        k=random(2);
        nacen++;
        if (k==0) {
            hijo.datos.sexo='M';
            agregar_mujer(hijo);
            nmujeres++;
        }
        else {
            hijo.datos.sexo='H';
            agregar_hombre(hijo);
        }
    }
}

```

```

                                nhombres++;
                                }//else
                                auxH=auxH->sig;
                                auxM=auxM->sig;
                                }//while
                                }//f
                                }

void AGpob::crossoverU(void) {
    poblacion *auxH,*auxM;
    persona hijo;
    double U;
    char hijos[14][longicromo+1]; //posibles hijos matematicos
    int i,pos,k1,k2,k;
        if (limites==1) pos=longicromo-1;
        else pos=1+random(longicromo-1);
        auxH=Hpob;
        auxM=Mpob;
        while (auxH!=NULL&&auxM!=NULL&&auxH->info.gene.cromosoma2[0]!=NULL&&auxM-
>info.gene.cromosoma2[0]!=NULL) {
            for (i=0;i<longicromo;i++) {
                hijos[0][i]=conjuncion(auxH->info.gene.cromosoma2[i],auxM-
>info.gene.cromosoma2[i]);
                hijos[1][i]=disyuncion(auxH->info.gene.cromosoma2[i],auxM-
>info.gene.cromosoma2[i]);
                hijos[2][i]=conjuncion(no(auxH->info.gene.cromosoma2[i]),no(auxM-
>info.gene.cromosoma2[i]));
                hijos[3][i]=disyuncion(no(auxH->info.gene.cromosoma2[i]),no(auxM-
>info.gene.cromosoma2[i]));
                hijos[4][i]=conjuncion(auxH->info.gene.cromosoma2[i],no(auxM-
>info.gene.cromosoma2[i]));
                hijos[5][i]=disyuncion(auxH->info.gene.cromosoma2[i],no(auxM-
>info.gene.cromosoma2[i]));
                hijos[6][i]=conjuncion(no(auxH->info.gene.cromosoma2[i]),auxM-
>info.gene.cromosoma2[i]);
                hijos[7][i]=disyuncion(no(auxH->info.gene.cromosoma2[i]),auxM-
>info.gene.cromosoma2[i]);
                hijos[8][i]= igual (auxH->info.gene.cromosoma2[i],auxM-
>info.gene.cromosoma2[i]);
                hijos[9][i]=diferente(auxH->info.gene.cromosoma2[i],auxM-
>info.gene.cromosoma2[i]);
                hijos[10][i]=auxH->info.gene.cromosoma2[i];
                hijos[11][i]=auxM->info.gene.cromosoma2[i];
                hijos[12][i]=no(auxH->info.gene.cromosoma2[i]);
                hijos[13][i]=no(auxM->info.gene.cromosoma2[i]);
            }//for
            for (i=0;i<14;i++) hijos[i][longicromo]=NULL;
            U=U01();
            if (Elitismo==0||auxH->info.gene.aptitud<mayaptitud&&U<=Pcro) {
                k1=(int)random(14);
                k2=(int)random(14);
            }//f
            else {k1=10;k2=11;}//else
            for (i=0;i<pos;i++) hijo.gene.cromosoma[i]=hijos[k1][i];
            for (i=pos;i<longicromo;i++) hijo.gene.cromosoma[i]=hijos[k2][i];
            hijo.gene.cromosoma[longicromo]=NULL;
            hijo.gene.cromosoma2[0]=NULL;
            hijo.datos.edad_actual=0;
            hijo.datos.edad_limite=muerte(muere);
        }
    }
}

```

```

do {
    hijo.datos.num_tot_hijos=(int)normal(nhijos,1);
} while (hijo.datos.num_tot_hijos<0);
hijo.datos.num_hijos=0;
do {
    hijo.datos.reproa=(int)nacea+random(10);
    hijo.datos.reprob=(int)naceb-random(10);
} while (hijo.datos.reproa>=hijo.datos.reprob);
k=random(2);
nacen++;
if (k==0) {
    hijo.datos.sexo='M';
    agregar_mujer(hijo);
    nmujeres++;
}
else {
    hijo.datos.sexo='H';
    agregar_hombre(hijo);
    nhombres++;
}
else
    auxH=auxH->sig;
    auxM=auxM->sig;
}
}

void AGpob::mutacion1(int x) {
    poblacion *aux;
    double U;
    int pos;
    if (exigir==1) U=1;
    else U=U01();
    if (U<=Pmut) {
        pos=1+random(longicromo-1);
        if (x==0) aux=Mpob;
        else aux=Hpob;
        while (aux!=NULL&&aux->sig!=NULL) {
            if (aux->info.gene.aptitud<=proaptitud) {
                if (aux->info.gene.cromosoma[pos]=='1') aux-
>info.gene.cromosoma[pos]='0';
                else aux->info.gene.cromosoma[pos]='1';
            }
            aux=aux->sig;
        }
    }
}

void AGpob::mutacionU(int x) {
    poblacion *aux;
    double U;
    int pos;
    pos=1+random(longicromo-1);
    if (x==0) aux=Mpob;
    else aux=Hpob;
    while (aux!=NULL&&aux->sig!=NULL) {
        if (exigir==1) U=1;
        else U=U01();
        if (aux->info.gene.aptitud<=proaptitud&&U<=Pmut) {
            if (aux->info.gene.cromosoma[pos]=='1') aux-
>info.gene.cromosoma[pos]='0';

```

```

        else aux->info.gene.cromosoma[pos]='1';
    }
    aux=aux->sig;
}

void AGpob::pasa_tiempo(void) {
    poblacion *aux;
    aux=Hpob;
    while (aux!=NULL) {
        aux->info.datos.edad_actual++;
        aux=aux->sig;
    }
    aux=Mpob;
    while (aux!=NULL) {
        aux->info.datos.edad_actual++;
        aux=aux->sig;
    }
}

void AGpob::elimina_viejos(void) {
    poblacion *aux;
    aux=Hpob;
    while (aux!=NULL) {
        if (aux->info.datos.edad_actual>=aux->info.datos.edad_limite) {
            quitar_primeroh();
            nhombres--;
            mueren++;
            if (Hpob!=NULL) aux=Hpob;
            else break;
        }
        else break;
        aux=aux->sig;
    }
    aux=Mpob;
    while (aux!=NULL) {
        if (aux->info.datos.edad_actual>=aux->info.datos.edad_limite) {
            quitar_primerom();
            nmujeres--;
            mueren++;
            if (Mpob!=NULL) aux=Mpob;
            else break;
        }
        else break;
        aux=aux->sig;
    }
}

double AGpob::genotipo(char A[]) {
    int j,k;
    double suma;
    if (A[0]!='0') k=0;
    else k=1;
    suma=(double)k;
    for (j=1;j<longicromo;j++) {
        suma*=2;
        if (A[j]!='0') k=0;
        else k=1;
        suma+=(double)k;
    }
}

```



```

        } //for
        return suma;
    }

    void AGpob::reemplazo(void) {
        poblacion *aux;
        aux=Hpob;
        while (aux!=NULL) {
            aux->info.gene.genotipo=genotipo(aux->info.gene.cromosoma)/(nhombres+nmujeres);
            aux->info.gene.aptitud=aptitud(aux->info.gene.genotipo)*puede_reproducirse(aux);
            aux=aux->sig;
        } //while
        aux=Mpob;
        while (aux!=NULL) {
            aux->info.gene.genotipo=genotipo(aux->info.gene.cromosoma)/(nhombres+nmujeres);
            aux->info.gene.aptitud=aptitud(aux->info.gene.genotipo)*puede_reproducirse(aux);
            aux=aux->sig;
        } //while
    }

    void AGpob::halla_mayproaptitud(void) {
        poblacion *aux;
        mayaptitud=0.0;
        proaptitud=0.0;
        proedad=0.0;
        proedadh=0.0;
        proedadm=0.0;
        aux=Hpob;
        while (aux!=NULL) {
            if (aux->info.gene.aptitud>mayaptitud) mayaptitud=aux->info.gene.aptitud;
            proaptitud+=aux->info.gene.aptitud;
            proedad+=aux->info.datos.edad_actual;
            proedadh+=aux->info.datos.edad_actual;
            aux=aux->sig;
        } //while
        aux=Mpob;
        while (aux!=NULL) {
            if (aux->info.gene.aptitud>mayaptitud) mayaptitud=aux->info.gene.aptitud;
            proaptitud+=aux->info.gene.aptitud;
            proedad+=aux->info.datos.edad_actual;
            proedadm+=aux->info.datos.edad_actual;
            aux=aux->sig;
        } //while
        if (nhombres!=0||nmujeres!=0) {
            proaptitud/=(nhombres+nmujeres);
            proedad/=(nhombres+nmujeres);
            if (nhombres!=0) proedadh/=(double)nhombres;
            else proedadh=0.0;
            if (nmujeres!=0) proedadm/=(double)nmujeres;
            else proedadm=0.0;
        } //if
        else {
            proaptitud=0.0;
            proedad=0.0;
        } //else
    }

    void AGpob::guardar_resultados(int x) {
        FILE *qtr;

```

```
long nmr=num_reproduce(0);
```

```
qtr=fopen("Resul1.Agr","at");  
if (x==0) fprintf(qtr,"%0lf %0lf %0lf\n",nhombres,nmujeres,nhombres+nmujeres);  
else fprintf(qtr,"%0lf %0lf %0lf",nhombres,nmujeres,nhombres+nmujeres);  
fclose(qtr);
```

```
qtr=fopen("Resul2.Agr","at");  
if (nanterior!=-1) {  
    crecimiento=(double)((nhombres+nmujeres)-nanterior)/nanterior;  
    crecih=(double)(nhombres-nanterior)/nanterior;  
    crecim=(double)(nmujeres-nanterior)/nanterior;
```

```
    }  
    //if  
    else {
```

```
        crecimiento=0.0;  
        crecih=0.0;  
        crecim=0.0;
```

```
    }  
    //else
```

```
    if (x==0) fprintf(qtr," %lf %lf %lf\n",crecih*1000.0,crecim*1000.0,crecimiento*1000.0);  
    else fprintf(qtr," %lf %lf %lf",crecih*1000.0,crecim*1000.0,crecimiento*1000.0);  
    fclose(qtr);
```

```
    if (nanterior==--1) mueren=0;
```

```
    qtr=fopen("Resul6.Agr","at");
```

```
    if (x==0) fprintf(qtr,"%0lf %0lf
```

```
%0lf\n",((double)(nacen*1000.0/nanterior),((double)(mueren*1000.0/nanterior),((double)((nacen-  
mueren)*1000.0/nanterior));
```

```
    else fprintf(qtr,"%0lf %0lf
```

```
%0lf",((double)(nacen*1000.0/nanterior),((double)(mueren*1000.0/nanterior),((double)((nacen-  
mueren)*1000.0/nanterior));
```

```
    fclose(qtr);
```

```
qtr=fopen("Resul3.Agr","at");
```

```
fprintf(qtr,"%lf ",mayaptitud);
```

```
if (x==0) fprintf(qtr,"%lf\n",proaptitud);
```

```
else fprintf(qtr,"%lf",proaptitud);
```

```
fclose(qtr);
```

```
qtr=fopen("Resul4.Agr","at");
```

```
if (x==0) fprintf(qtr,"%02lf %02lf %02lf\n",proedadh,proedadm,proedad);
```

```
else fprintf(qtr,"%02lf %02lf %02lf",proedadh,proedadm,proedad);
```

```
fclose(qtr);
```

```
hanterior=nhombres;
```

```
manterior=nmujeres;
```

```
nanterior=nhombres+nmujeres;
```

```
qtr=fopen("AgData.Rst","at");
```

```
if (x==0) fprintf(qtr,"%0lf\n",nhombres+nmujeres);
```

```
else fprintf(qtr,"%0lf",nhombres+nmujeres);
```

```
fclose(qtr);
```

```
qtr=fopen("AgData2.Rst","at");
```

```
if (x==0) fprintf(qtr,"%0lf %ld %ld %0lf
```

```
%0lf\n",nhombres+nmujeres,nacen,mueren,nacen*proaptitud,mueren*proaptitud);
```

```
else fprintf(qtr,"%0lf %ld %ld %0lf
```

```
%0lf",nhombres+nmujeres,nacen,mueren,nacen*proaptitud,mueren*proaptitud);
```

```
fclose(qtr);
```

```
qtr=fopen("Resul7.Agr","at");
```

```

        fprintf(qtr,"%lf %lf\n",
        (double)(nhombres*1000.0/nanterior),(double)(nmujeres*1000.0/nanterior));
        if (x==0) fprintf(qtr,"%lf\n", (double)(nmr*1000.0/nmujeres));
        else fprintf(qtr,"%lf", (double)(nmr*1000.0/nmujeres));
        fclose(qtr);

        nacen=0;
        mueren=0;
    }

    void AGpob::crear(void) {
        Hpob=NULL;
        Mpob=NULL;
    }

    void AGpob::destruir(void) {
        FILE *qtr;
        int gente1[50];
        int gente2[50];
        long tempo=(long)ceil((double)(muere/5));
        poblacion *aux;
        aux=Hpob;
        for (int i=0;i<tempo;i++) {
            gente1[i]=0;
            gente2[i]=0;
        }
        while (aux!=NULL) {
            for (int i=0;i<tempo;i++) {
                if (aux->info.datos.edad_actual<=(i+1)*5+1) {
                    gente1[i]++;
                    break;
                }
            }
            aux=aux->sig;
        }
        aux=Mpob;
        while (aux!=NULL) {
            for (int i=0;i<tempo;i++) {
                if (aux->info.datos.edad_actual<=(i+1)*5+1) {
                    gente2[i]++;
                    break;
                }
            }
            aux=aux->sig;
        }
        DeleteFile("Resul5.Agr");
        qtr=fopen("Resul5.Agr","at");
        for (i=0;i<tempo;i++) fprintf(qtr,"%d\n",gente1[i]);
        for (i=0;i<tempo;i++) fprintf(qtr,"%d\n",gente2[i]);
        fclose(qtr);
        while (Hpob!=NULL) quitar_primeroh();
        while (Mpob!=NULL) quitar_primerom();
    }

    void AGpob::quitar_primerom(void) {
        poblacion *R;
        if (Mpob!=NULL) {
            R=Mpob;
            Mpob=Mpob->sig;
        }
    }

```

```

        delete R;
    }//f
}

void AGpob::quitar_primeroh(void) {
    poblacion *R;
    if(Hpob!=NULL) {
        R=Hpob;
        Hpob=Hpob->sig;
        delete R;
    }//f
}

void AGpob::agregar_hombre(persona x) {
    poblacion *R;
    if (Hpob==NULL) {
        Hpob=(poblacion *)malloc(sizeof(poblacion));
        Hpob->sig=NULL;
        Hpob->info=x;
    }//f
    else {
        R=Hpob;
        while (R->sig!=NULL) R=R->sig;
        R->sig=(poblacion *)malloc(sizeof(poblacion));
        R=R->sig;
        R->sig=NULL;
        R->info=x;
    }//else
}

void AGpob::agregar_mujer(persona x) {
    poblacion *R;
    if (Mpob==NULL) {
        Mpob=(poblacion *)malloc(sizeof(poblacion));
        Mpob->sig=NULL;
        Mpob->info=x;
    }//f
    else {
        R=Mpob;
        while (R->sig!=NULL) R=R->sig;
        R->sig=(poblacion *)malloc(sizeof(poblacion));
        R=R->sig;
        R->sig=NULL;
        R->info=x;
    }//else
}

int AGpob::puede_reproducirse(poblacion *pob) {
    if (pob->info.datos.edad_actual>=pob->info.datos.reproa) {
        if (pob->info.datos.edad_actual<=pob->info.datos.reprob) {
            if (pob->info.datos.num_hijos<pob->info.datos.num_tot_hijos) return 1;
        }//f
    }//f
    return 0;
}

void AGpob::ordenar(int ff,int x) {
    poblacion *temp,*aux;
    persona aux2;

```



```

if (x==0) aux=Hpob;
else aux=Mpob;
    if (ff==0&&aux!=NULL) { //ordena de acuerdo a la edad de mayor a menor
        while (aux->sig!=NULL) {
            temp=aux->sig;
            while (temp!=NULL) {
                if (temp->info.datos.edad_limite-temp->info.datos.edad_actual<aux-
>info.datos.edad_limite-aux->info.datos.edad_actual) {
                    aux2=temp->info;
                    temp->info=aux->info;
                    aux->info=aux2;
                }
            }
            temp=temp->sig;
        }
        aux=aux->sig;
    }
}

//if
else if (ff==1&&aux!=NULL) { //ordena de acuerdo a la aptitud de mayor a menor
    while (aux->sig!=NULL) {
        temp=aux->sig;
        while (temp!=NULL) {
            if (temp->info.gene.aptitud>aux->info.gene.aptitud) {
                aux2=temp->info;
                temp->info=aux->info;
                aux->info=aux2;
            }
        }
        temp=temp->sig;
    }
    aux=aux->sig;
}

//while
}

//else if
else if (ff==2&&aux!=NULL) { //ordena de acuerdo a la reproduccion
    while (aux->sig!=NULL) {
        temp=aux->sig;
        while (temp!=NULL) {
            if (puede_reproducirse(temp)&&!puede_reproducirse(aux)) {
                aux2=temp->info;
                temp->info=aux->info;
                aux->info=aux2;
            }
        }
        temp=temp->sig;
    }
    aux=aux->sig;
}

//while
}

//else
else if (ff==3&&aux!=NULL) { //ordena de acuerdo a la seleccion
    while (aux->sig!=NULL) {
        temp=aux->sig;
        while (temp!=NULL) {
            if (aux->info.gene.cromosoma2[0]==NULL&&temp-
>info.gene.cromosoma2[0]!=NULL) {
                aux2=temp->info;
                temp->info=aux->info;
                aux->info=aux2;
            }
        }
        temp=temp->sig;
    }
    aux=aux->sig;
}

//while
}

```

```

        } //if
    }

    long AGpob::num_reproduce(int x) {
        poblacion *aux;
        long contador=0;
        if (x==0) aux=Mpob;
        else aux=Hpob;
        while (aux!=NULL) {
            if (puede_reproducirse(aux)) contador++;
            aux=aux->sig;
        } //while
        return contador;
    }

    double AGpob::normal(double mu,double s2) {
        double c,v1,v2,u1,u2,u;
        int x;
        do {
            u=(float)rand()/RAND_MAX;
            u1=(float)rand()/RAND_MAX;
            u2=(float)rand()/RAND_MAX;
            v1=2*u1-1;
            v2=2*u2-1;
        } while(v1*v1+v2*v2>1);
        c=sqrt(-2*log(u)/u);
        x=random(2);
        if (x==0) return (v1*c)*s2+mu;
        else return (v2*c)*s2+mu;
    }
}

```

Archivo Resulta.cpp

Resultados de la Simulación

```
#include "stdafx.h"
#include "math.h"
#include "string.h"
#include "AgMp.h"

#include "ProgressWnd.h"
#include "Agpob.hpp"
#include "RoundButton.h"
#include "Guardar.h"

#include "Resulta.h"
#include "GraEdades.h"
#include "Graapti1.h"
#include "Gracreci1.h"
#include "Gracrecimiento1.h"
#include "TBN.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

AGpob G;

CResulta::CResulta(CWnd* pParent /*=NULL*/)
: CDialog(CResulta::IDD, pParent){
//{{AFX_DATA_INIT(CResulta)
m_aptitudes = _T("");
m_crecimiento = _T("");
m_archivo = _T("");
//}}AFX_DATA_INIT
}

void CResulta::DoDataExchange(CDataExchange* pDX) {
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CResulta)
DDX_Control(pDX, IDC_Edades, m_edades);
DDX_Control(pDX, IDC_Guardar, m_Guardar);
DDX_Control(pDX, IDOK, m_ok);
DDX_Control(pDX, IDC_Gra2, m_gra2);
DDX_Control(pDX, IDC_Gra3, m_gra3);
DDX_Control(pDX, IDC_Imprimir, m_impri);
DDX_Control(pDX, IDC_Gra1, m_gra1);
DDX_Text(pDX, IDC_aptitudes, m_aptitudes);
DDX_Text(pDX, IDC_crecimiento, m_crecimiento);
DDX_Text(pDX, IDC_Archivo, m_archivo);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CResulta, CDialog)
//{{AFX_MSG_MAP(CResulta)
ON_BN_CLICKED(IDC_Gra1, OnGra1)
```

```

ON_BN_CLICKED(IDC_Gra2, OnGra2)
ON_BN_CLICKED(IDC_Gra3, OnGra3)
ON_BN_CLICKED(IDC_Imprimir, OnImprimir)
ON_WM_SHOWWINDOW()
ON_BN_CLICKED(IDC_Guardar, OnGuardar)
ON_BN_CLICKED(IDC_Edades, OnEdades)
ON_COMMAND(IDC_Tasas, OnTasas)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CResulta::Proceso(void) {
FILE *qtr;
long nhr,nmr;
    qtr=fopen("confi.Agc","rt");
    if (qtr!=NULL) {
        fclose(qtr);
        DeleteFile("Resul1.Agr");
        DeleteFile("Resul2.Agr");
        DeleteFile("Resul3.Agr");
        DeleteFile("Resul4.Agr");
        DeleteFile("Resul5.Agr");
        DeleteFile("Resul6.Agr");
        DeleteFile("Resul7.Agr");
        DeleteFile("AgData.Rst");
        DeleteFile("AgData2.Rst");
        G.generainicial();//Población inicial
        CProgressWnd wndProgress(this, "Procesando el Algoritmo Genético", TRUE);
        wndProgress.SetRange(0,G._genera());
        wndProgress.SetText("Por favor, Espere un momento...");
        while (G.converge()==0) {
            wndProgress.StepIt();
            wndProgress.PeekAndPump();
            if (wndProgress.Cancelled()) return FALSE;
            G.ordenar(0,0);G.ordenar(0,1);
            G.pasa_tiempo();
            G.elimina_viejos();
            G.reemplazo();
            G.halla_apitud01();
            G.halla_mayproaptitud();
            G.ordenar(2,0);G.ordenar(2,1);
            G.guardar_resultados(0);
            if (G.Logistica_permite()==1) {
                nmr=G.num_reproduce(0);
                nhr=G.num_reproduce(1);
                if (G._sele()==0)
                    {G.seleccion_Ruleta(0,nmr);G.seleccion_Ruleta(1,nhr);}//if
                else G.seleccion_Torneo(nhr,nmr);
                G.ordenar(3,0);G.ordenar(3,1);
                if (G._tipocros()==0) G.crossover1();
                else G.crossoverU();
            }//if
            if (G._tipomuta()==0) {G.mutacion1(0);G.mutacion1(1);}//if
            else {G.mutacionU(0);G.mutacionU(1);}//else
        }//while
        G.ordenar(1,0);G.ordenar(1,1);
        G.guardar_resultados(1);
        G.destruir();
    }//if
}

```

```

        else MessageBox("Por favor cree una Configuración antes de Ejecutar", "Ejecutar el Algoritmo Genético");
        return TRUE;
}

```

```

void CResulta::Muestra(void) {
FILE *qtr,*ptr;
char aux[20];
int i=0;

ptr=fopen("Resul1.Agr","rt");
qtr=fopen("Resul2.Agr","rt");
rewind(qtr);
while(!feof(qtr)) {
    itoa(i,aux,10);
    if (i<10) m_crecimiento+="0";
    m_crecimiento+=aux;
    m_crecimiento+="\t";
    fscanf(ptr,"%s",&aux);
    m_crecimiento+=aux;
    m_crecimiento+="\t";
    fscanf(ptr,"%s",&aux);
    m_crecimiento+=aux;
    m_crecimiento+="\t";
    fscanf(ptr,"%s",&aux);
    m_crecimiento+=aux;
    m_crecimiento+="\t";

    fscanf(qtr,"%s",&aux);
    fscanf(qtr,"%s",&aux);
    fscanf(qtr,"%s",&aux);
    if (atoi(aux)>=0) m_crecimiento+=" ";
    m_crecimiento+=aux;
    m_crecimiento+=13;
    m_crecimiento+=10;
    i++;
}

fclose(ptr);
fclose(qtr);
i=0;
ptr=fopen("Resul3.Agr","rt");
qtr=fopen("Resul4.Agr","rt");
rewind(ptr);rewind(qtr);
while(!feof(qtr)) {
    itoa(i,aux,10);
    if (i<10) m_aptitudes+="0";
    m_aptitudes+=aux;
    m_aptitudes+="\t";
    fscanf(ptr,"%s",&aux);
    m_aptitudes+=aux;
    m_aptitudes+=" ";
    m_aptitudes+="\t";
    fscanf(ptr,"%s",&aux);
    m_aptitudes+=aux;
    m_aptitudes+=" ";
    m_aptitudes+="\t";

    fscanf(qtr,"%s",&aux);
    fscanf(qtr,"%s",&aux);
    fscanf(qtr,"%s",&aux);
}
}

```



```

        m_apitudes+=aux;
        m_apitudes+=13;
        m_apitudes+=10;
        i++;
    }//while
    fclose(ptr);
    fclose(qtr);
    UpdateData(FALSE);
}

void CResulta::OnShowWindow(BOOL bShow, UINT nStatus) {
    CDialog::OnShowWindow(bShow, nStatus);
    Muestra();
}

void CResulta::Imprime_resultados(void) {
    int i=0;
    FILE *qtr;
    char aux[30];
    m_archivo+="\n\n-----";
    m_archivo+="\n « Resultados del AGMP »";
    m_archivo+="\n-----";
    m_archivo+="\nAño";
    m_archivo+="\t";m_archivo+="Hombres";
    m_archivo+="\t";m_archivo+="Mujeres";
    m_archivo+="\t";m_archivo+="Total";
    m_archivo+="\n";
    qtr=fopen("Resul1.Agr","rt");
    rewind(qtr);
    if (qtr!=NULL) {
        while(!feof(qtr)) {
            m_archivo+=" ";itoa(i,aux,10);
            if (atoi(aux)<10) m_archivo+="0";
            m_archivo+=aux;
            fscanf(qtr,"%s",&aux);
            m_archivo+="\t";m_archivo+=aux;
            fscanf(qtr,"%s",&aux);
            m_archivo+="\t";m_archivo+=aux;
            fscanf(qtr,"%s",&aux);
            m_archivo+="\t";m_archivo+=aux;
            m_archivo+="\n";
            i++;
        }//while
        fclose(qtr);
    }//if

    i=0;
    m_archivo+="\n\nAño";
    m_archivo+="\t";m_archivo+="H_Creci";
    m_archivo+="\t\t";m_archivo+="M_Creci";
    m_archivo+="\t\t";m_archivo+="T_Creci";
    m_archivo+="\n";
    qtr=fopen("Resul2.Agr","rt");
    rewind(qtr);
    if (qtr!=NULL) {
        while(!feof(qtr)) {
            m_archivo+=" ";itoa(i,aux,10);
            if (atoi(aux)<10) m_archivo+="0";
            m_archivo+=aux;

```

```

        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        m_archivo+="\n";
        i++;
    }//while
    fclose(qtr);
}

i=0;
m_archivo+="\n\nAño";
m_archivo+="\t";m_archivo+="M_Aptitud";
m_archivo+="\t";m_archivo+="Aptitud_P";
m_archivo+="\n";
qtr=fopen("Resul3.Agr","rt");
rewind(qtr);
if (qtr!=NULL) {
    while(!feof(qtr)) {
        m_archivo+=" ";itoa(i,aux,10);
        if (atof(aux)<10) m_archivo+="0";
        m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        m_archivo+="\n";
        i++;
    }//while
    fclose(qtr);
}

i=0;
m_archivo+="\n\nAño";
m_archivo+="\t";m_archivo+="E_P_H";
m_archivo+="\t";m_archivo+="E_P_M";
m_archivo+="\t";m_archivo+="Edad_P";
m_archivo+="\n";
qtr=fopen("Resul4.Agr","rt");
rewind(qtr);
if (qtr!=NULL) {
    while(!feof(qtr)) {
        m_archivo+=" ";itoa(i,aux,10);
        if (atof(aux)<10) m_archivo+="0";
        m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        m_archivo+="\n";
        i++;
    }//while
    fclose(qtr);
}

```

```

}

void CResulta::PrintString(const char *p, DWORD pSize) {
    CDC      dc;
    CPrintDialog  printDlg(FALSE);
    CRect      r;
    int      nHeight;
    if (printDlg.DoModal() == IDCANCEL) return;
    dc.Attach(printDlg.GetPrinterDC());
    dc.m_bPrinting = TRUE;
    dc.SetMapMode(MM_TEXT);
    LOGFONT LogFont;
    CFont aFont, *oldFont;
    LogFont.lfHeight = -MulDiv(10, GetDeviceCaps(dc, LOGPIXELSY), 72);
    LogFont.lfWidth = 0;
    LogFont.lfEscapement = 0;
    LogFont.lfOrientation = 0;
    LogFont.lfWeight = 0;
    LogFont.lfItalic = false;
    LogFont.lfUnderline = 0;
    LogFont.lfStrikeOut = 0;
    LogFont.lfCharSet = ANSI_CHARSET;
    LogFont.lfOutPrecision = OUT_TT_PRECIS;
    LogFont.lfClipPrecision = CLIP_DEFAULT_PRECIS;
    LogFont.lfQuality = DEFAULT_QUALITY;
    LogFont.lfPitchAndFamily = DEFAULT_PITCH | FF_SWISS;
    lstrcpy(LogFont.lfFaceName, "MS Sans Serif");
    dc.SetBkMode(OPAQUE);
    aFont.CreateFontIndirect(&LogFont);
    oldFont = dc.SelectObject(&aFont);
    CString strTitle;
    strTitle.LoadString(AFX_IDS_APP_TITLE);
    DOCINFO di;
    ::ZeroMemory(&di, sizeof(DOCINFO));
    di.cbSize = sizeof(DOCINFO);
    di.lpszDocName = strTitle;
    BOOL bPrintingOK = dc.StartDoc(&di);
    CPrintInfo Info;
    int w = dc.GetDeviceCaps(HORZRES);
    int h = dc.GetDeviceCaps(VERTRES);
    Info.m_rectDraw.SetRect(0,0, w, h);
    char *startAt=(char *)p;
    int totalDone = 0;
    int lengthToGo = pSize;
    for (UINT page = Info.GetMinPage(); bPrintingOK && totalDone < lengthToGo; page++) {
        dc.StartPage();
        Info.m_nCurPage = page;
        r = Info.m_rectDraw;
        r.bottom = r.top;
        int i = 0;
        while (r.bottom < Info.m_rectDraw.bottom && totalDone + i < lengthToGo) {
            Info.m_rectDraw.right;
            nHeight = dc.DrawText(startAt, i++,
r, DT_CALCRECT|DT_WORDBREAK|DT_NOCLIP|DT_EXPANDTABS);
            //while
            if (r.bottom >= Info.m_rectDraw.bottom) i--;
            dc.DrawText(startAt, i, r, DT_WORDBREAK|DT_NOCLIP|DT_EXPANDTABS);
            startAt += i;
            totalDone += i;
        }
    }
}

```



```

        bPrintingOK = (dc.EndPage() > 0);
    } //for
    if (bPrintingOK) dc.EndDoc();
    else dc.AbortDoc();
    dc.SelectObject(oldFont);
    aFont.DeleteObject();
    dc.Detach();
}

void CResulta::OnOK(){CDialog::OnOK();}

void CResulta::OnGra1() {
    CGraapti Gr1;
    Gr1.DoModal();
}

void CResulta::OnGra2() {
    CGracreci Gr2;
    Gr2.DoModal();
}

void CResulta::OnGra3() {
    CGracrecimiento Gr3;
    Gr3.DoModal();
}

void CResulta::OnImprimir() {
    Imprime_resultados();
    PrintString(m_archivo,m_archivo.GetLength());
}

void CResulta::OnGuardar() {
    CGuardar Gu;
    Gu.DoModal();
}

void CResulta::OnEdades() {
    CGraEdades Gr4;
    Gr4.DoModal();
}

void CResulta::OnTasas() {
    CTBN TN;
    TN.DoModal();
}

```

Archivo de encabezado Resulta.h

```
#include "RoundButton.h"
#ifdef AFX_RESULTA_H__0CF52401_2C22_11D5_AB39_E68BF3988861__INCLUDED_
#define AFX_RESULTA_H__0CF52401_2C22_11D5_AB39_E68BF3988861__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CResulta : public CDialog{
// Construction
public:
    CResulta(CWnd* pParent = NULL); // standard constructor
    void PrintString(const char *p, DWORD pSize);
    void Imprime_resultados(void);
    void Muestra(void);
    BOOL Proceso(void);

// Dialog Data
    //{AFX_DATA(CResulta)
    enum { IDD = IDD_RESULTA_DIALOG };
    CRoundButton m_edades;
    CRoundButton m_Guardar;
    CRoundButton m_ok;
    CRoundButton m_gra2;
    CRoundButton m_gra3;
    CRoundButton m_impri;
    CRoundButton m_gra1;
    CString m_apitudes;
    CString m_crecimiento;
    CString m_archivo;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CResulta)
    protected:
        virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(CResulta)
    afx_msg void OnGra1();
    afx_msg void OnGra2();
    afx_msg void OnGra3();
    afx_msg void OnImprimir();
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    virtual void OnOK();
    afx_msg void OnGuardar();
    afx_msg void OnEdades();
    afx_msg void OnTasas();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif // !defined(AFX_RESULTA_H__0CF52401_2C22_11D5_AB39_E68BF3988861__INCLUDED_)
```

Archivo TBN.cpp

```
#include "stdafx.h"
#include "AgMp.h"
#include "TBN.h"
#include "TBM.h"
#include "TBM2.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CTBN::CTBN(CWnd* pParent /*=NULL*/):CDialog(CTBN::IDD, pParent){
//{{AFX_DATA_INIT(CTBN)
    m_fecun = _T("");
    m_crecimiento = _T("");
    m_archivo = _T("");
//}}AFX_DATA_INIT
}

void CTBN::DoDataExchange(CDataExchange* pDX){
    CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CTBN)
    DDX_Control(pDX, IDOK, m_ok);
    DDX_Control(pDX, IDC_Imprimir, m_imprimir);
    DDX_Control(pDX, IDC_Gra3, m_gra3);
    DDX_Control(pDX, IDC_Fecun, m_fecundo);
    DDX_Text(pDX, IDC_fecundidad, m_fecun);
    DDX_Text(pDX, IDC_crecimiento, m_crecimiento);
    DDX_Text(pDX, IDC_Archivo, m_archivo);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CTBN, CDialog)
//{{AFX_MSG_MAP(CTBN)
    ON_WM_SHOWWINDOW()
    ON_BN_CLICKED(IDC_Gra3, OnGra3)
    ON_BN_CLICKED(IDC_Fecun, OnFecun)
    ON_BN_CLICKED(IDC_Imprimir, OnImprimir)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

int CTBN::DoModal() {return CDialog::DoModal();}

void CTBN::OnShowWindow(BOOL bShow, UINT nStatus) {
    FILE *ptr;
    char aux[20];
    int i=0;
    CDialog::OnShowWindow(bShow, nStatus);
    ptr=fopen("Resul6.Agr", "rt");
    rewind(ptr);
    while(!feof(ptr)) {
        itoa(i,aux,10);
        if (i<10) m_crecimiento+="0";
        m_crecimiento+=aux;
        m_crecimiento+=" ";
        i++;
    }
}
```

```

        fscanf(ptr,"%s",&aux);
        m_crecimiento+=aux;
        m_crecimiento+="\t";
        fscanf(ptr,"%s",&aux);
        m_crecimiento+=aux;
        m_crecimiento+="\t\t";
        fscanf(ptr,"%s",&aux);
        if (atof(aux)>=0) m_crecimiento+="+";
        m_crecimiento+=aux;
        m_crecimiento+=13;
        m_crecimiento+=10;
        i++;
    }//while
    fclose(ptr);
    i=0;
    ptr=fopen("Resul7.Agr","rt");
    rewind(ptr);
    while(!feof(ptr)) {
        itoa(i,aux,10);
        if (i<10) m_fecun+="0";
        m_fecun+=aux;
        m_fecun+="\t";
        fscanf(ptr,"%s",&aux);
        m_fecun+=aux;
        m_fecun+=" ";
        m_fecun+="\t";
        fscanf(ptr,"%s",&aux);
        m_fecun+=aux;
        m_fecun+=" ";
        m_fecun+="\t";
        fscanf(ptr,"%s",&aux);
        m_fecun+=aux;
        m_fecun+=13;
        m_fecun+=10;
        i++;
    }//while
    fclose(ptr);
    UpdateData(FALSE);
}

void CTBN::OnGra3() {
    CTBM TM;
    TM.DoModal();
}

void CTBN::OnFecun() {
    CTBM2 TM;
    TM.DoModal();
}

void CTBN::OnImprimir() {
    Imprime_resultados();
    PrintString(m_archivo,m_archivo.GetLength());
}

void CTBN::Imprime_resultados(void) {
    int i=0;
    FILE *qtr;
    char aux[30];

```

```

m_archivo+="\n\n-----";
m_archivo+="\n « Resultados del AGMP »";
m_archivo+="\n-----";
m_archivo+="\nAño";
m_archivo+="\t";m_archivo+="Masculinidad";
m_archivo+="\t";m_archivo+="Feminidad";
m_archivo+="\t";m_archivo+="Fecundidad";
m_archivo+="\n";
qtr=fopen("Resul7.Agr","rt");
rewind(qtr);
if (qtr!=NULL) {
    while(!feof(qtr)) {
        m_archivo+=" ";itoa(i,aux,10);
        if (atof(aux)<10) m_archivo+="0";
        m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        m_archivo+="\n";
        i++;
    }
    fclose(qtr);
}

m_archivo+="\n\nAño";
m_archivo+="\t";m_archivo+="Natalidad";
m_archivo+="\t";m_archivo+="Mortalidad";
m_archivo+="\t";m_archivo+="Crecimiento";
qtr=fopen("Resul6.Agr","rt");
rewind(qtr);
if (qtr!=NULL) {
    while(!feof(qtr)) {
        m_archivo+=" ";itoa(i,aux,10);
        if (atof(aux)<10) m_archivo+="0";
        m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";m_archivo+=aux;
        fscanf(qtr,"%s",&aux);
        m_archivo+="\t";
        if (atof(aux)>=0) m_archivo+=" ";
        m_archivo+=aux;
        m_archivo+="\n";
        i++;
    }
    fclose(qtr);
}
}

```

```

void CTBN::PrintString(const char *p, DWORD pSize) {
    CDC dc;
    CPrintDialog printDlg(FALSE);
    CRect r;
    int nHeight;
    if (printDlg.DoModal() == IDCANCEL) return;
}

```



```

dc.Attach(printDlg.GetPrinterDC());
dc.m_bPrinting = TRUE;
dc.SetMapMode(MM_TEXT);
LOGFONT LogFont;
CFont aFont, *oldFont;
LogFont.lfHeight = -MulDiv(10, GetDeviceCaps(dc, LOGPIXELSY), 72);
LogFont.lfWidth = 0;
LogFont.lfEscapement = 0;
LogFont.lfOrientation = 0;
LogFont.lfWeight = 0;
LogFont.lfItalic = false;
LogFont.lfUnderline = 0;
LogFont.lfStrikeOut = 0;
LogFont.lfCharSet = ANSI_CHARSET;
LogFont.lfOutPrecision = OUT_TT_PRECIS;
LogFont.lfClipPrecision = CLIP_DEFAULT_PRECIS;
LogFont.lfQuality = DEFAULT_QUALITY;
LogFont.lfPitchAndFamily = DEFAULT_PITCH | FF_SWISS;
lstrcpy(LogFont.lfFaceName, "MS Sans Serif");
dc.SetBkMode(OPAQUE);
aFont.CreateFontIndirect(&LogFont);
oldFont = dc.SelectObject(&aFont);
CString strTitle;
strTitle.LoadString(AFX_IDS_APP_TITLE);
DOCINFO di;
::ZeroMemory(&di, sizeof(DOCINFO));
di.cbSize = sizeof(DOCINFO);
di.lpszDocName = strTitle;
BOOL bPrintingOK = dc.StartDoc(&di);
CPrintInfo Info;
int w = dc.GetDeviceCaps(HORZRES);
int h = dc.GetDeviceCaps(VERTRES);
Info.m_rectDraw.SetRect(0, 0, w, h);
char *startAt = (char *)p;
int totalDone = 0;
int lengthToGo = pSize;
for (UINT page = Info.GetMinPage(); bPrintingOK && totalDone < lengthToGo; page++) {
    dc.StartPage();
    Info.m_nCurPage = page;
    r = Info.m_rectDraw;
    r.bottom = r.top;
    int i = 0;
    while (r.bottom < Info.m_rectDraw.bottom && totalDone + i < lengthToGo) {
        Info.m_rectDraw.right;
        nHeight = dc.DrawText(startAt, i++,
r, DT_CALCRECT | DT_WORDBREAK | DT_NOCLIP | DT_EXPANDTABS);
    } //while
    if (r.bottom >= Info.m_rectDraw.bottom) i--;
    dc.DrawText(startAt, i, r, DT_WORDBREAK | DT_NOCLIP | DT_EXPANDTABS);
    startAt += i;
    totalDone += i;
    bPrintingOK = (dc.EndPage() > 0);
} //for
if (bPrintingOK) dc.EndDoc();
else dc.AbortDoc();
dc.SelectObject(oldFont);
aFont.DeleteObject();
dc.Detach();
}

```

Archivo de encabezado TBN.h

```
#include "RoundButton.h"
#if !defined(AFX_TBN_H_9B4F0DA1_5D6F_11D5_AB3A_C13EF882B249__INCLUDED_)
#define AFX_TBN_H_9B4F0DA1_5D6F_11D5_AB3A_C13EF882B249__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CTBN : public CDialog {
// Construction
public:
    CTBN(CWnd* pParent = NULL); // standard constructor
    void Imprime_resultados(void);
    void PrintString(const char *p, DWORD pSize);
// Dialog Data
   //{{AFX_DATA(CTBN)
    enum { IDD = IDD_TBN_DIALOG };
    CRoundButton m_ok;
    CRoundButton m_imprimir;
    CRoundButton m_gra3;
    CRoundButton m_fecundo;
    CString m_fecun;
    CString m_crecimiento;
    CString m_archivo;
    //}}AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CTBN)
    public:
        virtual int DoModal();
    protected:
        virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(CTBN)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnGra3();
    afx_msg void OnFecun();
    afx_msg void OnImprimir();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
#endif // !defined(AFX_TBN_H_9B4F0DA1_5D6F_11D5_AB3A_C13EF882B249__INCLUDED_)
```

Archivo Gracreci1.cpp

Gráfico del Tamaño de la población

```
#include "stdafx.h"
#include "AgMp.h"
#include "Gracreci1.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CGracreci::CGracreci(CWnd* pParent /*=NULL*/): CDialog(CGracreci::IDD, pParent){
   //{{AFX_DATA_INIT(CGracreci)
    // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT
}

void CGracreci::DoDataExchange(CDataExchange* pDX){
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CGracreci)
    DDX_Control(pDX, IDC_Copiar, m_copiar);
    DDX_Control(pDX, IDOK, m_ok);
    DDX_Control(pDX, IDC_MSCHART1, m_Chart);
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CGracreci, CDialog)
    {{{AFX_MSG_MAP(CGracreci)
    ON_WM_SHOWWINDOW()
    ON_BN_CLICKED(IDC_Copiar, OnCopiar)
    }}}AFX_MSG_MAP
END_MESSAGE_MAP()

long CGracreci::numdatos(void) {
    long x;
    FILE *qtr;
    qtr=fopen("Confi.Agc","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        fscanf(qtr,"%ld",&x);
        fclose(qtr);
    }
    return x;
}

void CGracreci::OnShowWindow(BOOL bShow, UINT nStatus) {
    COleSafeArray saRet;
    FILE *qtr;
    long iter=0;
    long index[2];
    double val;
    char aux[20];
    CDialog::OnShowWindow(bShow, nStatus);
    iter=numdatos();
    DWORD numElements[] = {iter,3};
    saRet.Create(VT_R8, 2, numElements);
}
```



```

qtr=fopen("Resul1.Agr","rt");
for(index[0]=0;index[0]<iter; index[0]++) {
    index[1]=0;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
    index[1]=1;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
    index[1]=2;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
}
fclose(qtr);
m_Chart.SetChartData(saRet.Detach());
for (int i=0;i<iter;i++) {
    m_Chart.SetRow(i+1);
    itoa(i,aux,10);
    m_Chart.SetRowLabel(aux);
}
m_Chart.Refresh();
}

void CGracreci::OnCopiar() {m_Chart.EditCopy();}
void CGracreci::OnOK() {CDialog::OnOK();}

```

Archivo de encabezado Gracreci1.h

```
#include "RoundButton.h"
//{{AFX_INCLUDES()}
#include "mschart.h"
//}}AFX_INCLUDES
#if !defined(AFX_GRACRECI1_H__79C48062_2C29_11D5_AB39_E68BF3988861__INCLUDED_)
#define AFX_GRACRECI1_H__79C48062_2C29_11D5_AB39_E68BF3988861__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CGracreci : public CDialog{
// Construction
public:
    CGracreci(CWnd* pParent = NULL); // standard constructor
    long numdatos(void);

// Dialog Data
    //{AFX_DATA(CGracreci)
    enum { IDD = IDD_GRACRECI_DIALOG };
    CRoundButton    m_copiar;
    CRoundButton    m_ok;
    CMSChart        m_Chart;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CGracreci)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(CGracreci)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnCopiar();
    virtual void OnOK();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
!defined(AFX_GRACRECI1_H__79C48062_2C29_11D5_AB39_E68BF3988861__INCLUDED_)
```

Archivo Gracrecimiento1.cpp

Crecimiento de la población

```
#include "stdafx.h"
#include "AgMp.h"
#include "Gracrecimiento1.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CGracrecimiento::CGracrecimiento(CWnd* pParent /*=NULL*/): CDialog(CGracrecimiento::IDD,
pParent){
    ///{{AFX_DATA_INIT(CGracrecimiento)
    // NOTE: the ClassWizard will add member initialization here
    ///}}AFX_DATA_INIT
}

void CGracrecimiento::DoDataExchange(CDataExchange* pDX){
    CDialog::DoDataExchange(pDX);
    ///{{AFX_DATA_MAP(CGracrecimiento)
    DDX_Control(pDX, IDOK, m_ok);
    DDX_Control(pDX, IDC_Copiar, m_copiar);
    DDX_Control(pDX, IDC_MSCHART1, m_Chart);
    ///}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CGracrecimiento, CDialog)
    ///{{AFX_MSG_MAP(CGracrecimiento)
    ON_WM_SHOWWINDOW()
    ON_BN_CLICKED(IDC_Copiar, OnCopiar)
    ///}}AFX_MSG_MAP
END_MESSAGE_MAP()

long CGracrecimiento::numdatos(void) {
    long x;
    FILE *qtr;
    qtr=fopen("Confi.Agc","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        fscanf(qtr,"%ld",&x);
        fclose(qtr);
    }
    return x;
}

void CGracrecimiento::OnShowWindow(BOOL bShow, UINT nStatus) {
    COleSafeArray saRet;
    FILE *qtr;
    long iter=0;
    long index[2];
    double val;
    char aux[20];
    CDialog::OnShowWindow(bShow, nStatus);
    iter=numdatos();
    DWORD numElements[] = {iter,3};
```

```

saRet.Create(VT_R8, 2, numElements);
qtr=fopen("Resul2.Agr","rt");
for(index[0]=0;index[0]<iter; index[0]++) {
    index[1]=0;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
    index[1]=1;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
    index[1]=2;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
}
fclose(qtr);
m_Chart.SetChartData(saRet.Detach());
for (int i=0;i<iter;i++) {
    m_Chart.SetRow(i+1);
    itoa(i,aux,10);
    m_Chart.SetRowLabel(aux);
}
m_Chart.Refresh();
}

void CGracrecimiento::OnCopiar() {m_Chart.EditCopy();}
void CGracrecimiento::OnOK() {CDialog::OnOK();}

```

Archivo de encabezado Crecimiento1.h

```
#include "RoundButton.h"
//{{AFX_INCLUDES()
#include "mschart.h"
//}}AFX_INCLUDES
#if
!defined(AFX_GRACRECIMIENTO1_H__79C48063_2C29_11D5_AB39_E68BF3988861__INCLUDED_)
#define AFX_GRACRECIMIENTO1_H__79C48063_2C29_11D5_AB39_E68BF3988861__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CGracrecimiento : public CDialog{
// Construction
public:
    CGracrecimiento(CWnd* pParent = NULL); // standard constructor
    long numdatos(void);
// Dialog Data
    //{{AFX_DATA(CGracrecimiento)
    enum { IDD = IDD_GRACRECIMIENTO_DIALOG };
    CRoundButton    m_ok;
    CRoundButton    m_copiar;
    CMSChart        m_Chart;
    //}}AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CGracrecimiento)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG(CGracrecimiento)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnCopiar();
    virtual void OnOK();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
!defined(AFX_GRACRECIMIENTO1_H__79C48063_2C29_11D5_AB39_E68BF3988861__INCLUDED_)
_)
```


Archivo Graapti.cpp Gráfico de Aptitudes

```
#include "stdafx.h"
#include "AgMp.h"
#include "Graapti1.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CGraapti::CGraapti(CWnd* pParent /*=NULL*/): CDialog(CGraapti::IDD, pParent){
//{{AFX_DATA_INIT(CGraapti)
// NOTE: the ClassWizard will add member initialization here
//}}AFX_DATA_INIT
}

void CGraapti::DoDataExchange(CDataExchange* pDX) {
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CGraapti)
    DDX_Control(pDX, IDOK, m_ok);
    DDX_Control(pDX, IDC_Copiar, m_copiar);
    DDX_Control(pDX, IDC_MSCHART1, m_Chart);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CGraapti, CDialog)
//{{AFX_MSG_MAP(CGraapti)
ON_WM_SHOWWINDOW()
ON_BN_CLICKED(IDC_Copiar, OnCopiar)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

long CGraapti::numdatos(void) {
    long x;
    FILE *qtr;
    qtr=fopen("Confi.Agc","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        fscanf(qtr,"%ld",&x);
        fclose(qtr);
    }
    return x;
}

void CGraapti::OnShowWindow(BOOL bShow, UINT nStatus) {
    COleSafeArray saRet;
    FILE *qtr;
    long iter=0;
    long index[2];
    double val;
    char aux[20];
    CDialog::OnShowWindow(bShow, nStatus);
    iter=numdatos();
    DWORD numElements[] = {iter,2};
    saRet.Create(VT_R8, 2, numElements);
}
```

```

qtr=fopen("Resul3.Agr","rt");
for(index[0]=0;index[0]<iter; index[0]++) {
    index[1]=0;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
    index[1]=1;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
}
fclose(qtr);
m_Chart.SetChartData(saRet.Detach());
for (int i=0;i<iter;i++) {
    m_Chart.SetRow(i+1);
    itoa(i,aux,10);
    m_Chart.SetRowLabel(aux);
}
m_Chart.Refresh();
}

void CGraapti::OnCopiar() {m_Chart.EditCopy();}
void CGraapti::OnOK(){CDialog::OnOK();}

```

Archivo de encabezado Graapti.h

```
#include "RoundButton.h"
//{{AFX_INCLUDES()
#include "mschart.h"
//}}AFX_INCLUDES
#if !defined(AFX_GRAAPTI1_H__79C48061_2C29_11D5_AB39_E68BF3988861__INCLUDED_)
#define AFX_GRAAPTI1_H__79C48061_2C29_11D5_AB39_E68BF3988861__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CGraapti : public CDialog{
// Construction
public:
    CGraapti(CWnd* pParent = NULL); // standard constructor
    long numdatos(void);
// Dialog Data
    //{AFX_DATA(CGraapti)
    enum { IDD = IDD_GRAAPTI_DIALOG };
    CRoundButton    m_ok;
    CRoundButton    m_copiar;
    CMSChart        m_Chart;
    //}AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CGraapti)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL
// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(CGraapti)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnCopiar();
    virtual void OnOK();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif // !defined(AFX_GRAAPTI1_H__79C48061_2C29_11D5_AB39_E68BF3988861__INCLUDED_)
```


Archivo Graedades.h

Gráfico de edades

```
#include "stdafx.h"
#include "AgMp.h"
#include "GraEdades.h"
#include "Frecuencias.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CGraEdades::CGraEdades(CWnd* pParent /*=NULL*/): CDialog(CGraEdades::IDD, pParent){
//{{AFX_DATA_INIT(CGraEdades)
//}}AFX_DATA_INIT
}

void CGraEdades::DoDataExchange(CDataExchange* pDX){
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CGraEdades)
    DDX_Control(pDX, IDC_distrib, m_piramide);
    DDX_Control(pDX, IDC_Copiar, m_copiar);
    DDX_Control(pDX, IDOK, m_ok);
    DDX_Control(pDX, IDC_MSCHART1, m_Chart);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CGraEdades, CDialog)
    //{{AFX_MSG_MAP(CGraEdades)
    ON_BN_CLICKED(IDC_distrib, Ondistrib)
    ON_BN_CLICKED(IDC_Copiar, OnCopiar)
    ON_WM_SHOWWINDOW()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

long CGraEdades::numdatos(void) {
    long x;
    FILE *qtr;
    qtr=fopen("Confi.Agc","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        fscanf(qtr,"%ld",&x);
        fclose(qtr);
    }
    return x;
}

void CGraEdades::OnShowWindow(BOOL bShow, UINT nStatus) {
    COleSafeArray saRet;
    FILE *qtr;
    long iter=0;
    long index[2];
    double val;
    char aux[20];
    CDialog::OnShowWindow(bShow, nStatus);
    iter=numdatos();
}
```

```

DWORD numElements[] = {iter,3};
saRet.Create(VT_R8, 2, numElements);
qtr=fopen("Resul4.Agr","rt");
for(index[0]=0;index[0]<iter; index[0]++) {
    index[1]=0;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
    index[1]=1;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
    index[1]=2;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
}
fclose(qtr);
m_Chart.SetChartData(saRet.Detach());
for (int i=0;i<iter;i++) {
    m_Chart.SetRow(i+1);
    itoa(i,aux,10);
    m_Chart.SetRowLabel(aux);
}
m_Chart.Refresh();
}

void CGraEdades::OnCopiar() {m_Chart.EditCopy();}

void CGraEdades::OnOK() {CDialog::OnOK();}

void CGraEdades::Ondistribui() {
CFrecuencias Fr;
    Fr.DoModal();
}

```

Archivo de encabezado Graedades.h

```
#include "RoundButton.h"
//{{AFX_INCLUDES()
#include "mschart.h"
//}}AFX_INCLUDES
#if !defined(AFX_GRAEDADES_H__88379D82_573F_11D5_AB3A_871D99E85149__INCLUDED_)
#define AFX_GRAEDADES_H__88379D82_573F_11D5_AB3A_871D99E85149__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CGraEdades : public CDialog{
public:
    CGraEdades(CWnd* pParent = NULL);
    long numdatos(void);

// Dialog Data
    //{AFX_DATA(CGraEdades)
    enum { IDD = IDD_GRAEDADES_DIALOG };
    CRoundButton    m_piramide;
    CRoundButton    m_copiar;
    CRoundButton    m_ok;
    CMSChart        m_Chart;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CGraEdades)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(CGraEdades)
    afx_msg void Ondistribuir();
    afx_msg void OnCopiar();
    virtual void OnOK();
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
!defined(AFX_GRAEDADES_H__88379D82_573F_11D5_AB3A_871D99E85149__INCLUDED_)
```

Archivo Frecuencias.cpp

Gráfico de barras de la distribución por edades y sexo

```
#include "stdafx.h"
#include "AgMp.h"
#include "math.h"
#include "Frecuencias.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CFrecuencias::CFrecuencias(CWnd* pParent /*=NULL*/) : CDialog(CFrecuencias::IDD, pParent){
//{{AFX_DATA_INIT(CFrecuencias)
//}}AFX_DATA_INIT
}

void CFrecuencias::DoDataExchange(CDataExchange* pDX){
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CFrecuencias)
DDX_Control(pDX, IDOK, m_ok);
DDX_Control(pDX, IDC_Copiar, m_copiar);
DDX_Control(pDX, IDC_MSCHART1, m_Chart2);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CFrecuencias, CDialog)
//{{AFX_MSG_MAP(CFrecuencias)
ON_BN_CLICKED(IDC_Copiar, OnCopiar)
ON_WM_SHOWWINDOW()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

double CFrecuencias::numdatos(void) {
double x;
int i;
FILE *qtr;
qtr=fopen("Confi.Agc","rt");
if (qtr!=NULL) {
rewind(qtr);
for (i=0;i<16;i++) fscanf(qtr,"%lf",&x);
fclose(qtr);
}
return x;
}

void CFrecuencias::OnShowWindow(BOOL bShow, UINT nStatus) {
COleSafeArray saRet;
char aux[20],aux2[20];
FILE *qtr;
long index[2];
double val;
long iter=(long)ceil(numdatos()/5);
CDialog::OnShowWindow(bShow, nStatus);
DWORD numElements[] = {iter,2};
saRet.Create(VT_R8, 2, numElements);
```

```

qtr=fopen("Resul5.Agr","rt");
index[1]=0;
for(index[0]=0;index[0]<iter; index[0]++) {
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
}
index[1]=1;
for(index[0]=0;index[0]<iter; index[0]++) {
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
}
fclose(qtr);
m_Chart2.SetChartData(saRet.Detach());
m_Chart2.SetRow(1);
m_Chart2.SetRowLabel("0 - 5");
for (int i=1;i<iter;i++) {
    m_Chart2.SetRow(i+1);
    itoa(i*5+1,aux2,10);
    strcpy(aux,aux2);
    strcat(aux," - ");
    itoa((i+1)*5,aux2,10);
    strcat(aux,aux2);
    m_Chart2.SetRowLabel(aux);
}
m_Chart2.Refresh();
}

void CFrecuencias::OnOK() {CDialog::OnOK();}
void CFrecuencias::OnCopiar() {m_Chart2.EditCopy();}

```


Archivo de encabezado Frecuencias.h

```
#include "RoundButton.h"
//{{AFX_INCLUDES()
#include "mschart.h"
//}}AFX_INCLUDES
#ifndef AFX_FRECUENCIAS_H__88379D83_573F_11D5_AB3A_871D99E85149__INCLUDED_
#define AFX_FRECUENCIAS_H__88379D83_573F_11D5_AB3A_871D99E85149__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CFrecuencias : public CDialog {
// Construction
public:
    CFrecuencias(CWnd* pParent = NULL); // standard constructor
    double numdatos(void);

// Dialog Data
    {{{AFX_DATA(CFrecuencias)
    enum { IDD = IDD_FRECUENCIAS_DIALOG };
    CRoundButton    m_ok;
    CRoundButton    m_copiar;
    CMSChart        m_Chart2;
    }}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    {{{AFX_VIRTUAL(CFrecuencias)
    protected:
        virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    {{{AFX_MSG(CFrecuencias)
    virtual void OnOK();
    afx_msg void OnCopiar();
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
#endif AFX_FRECUENCIAS_H__88379D83_573F_11D5_AB3A_871D99E85149__INCLUDED_
```

Archivo Guardar.cpp

Guardar los datos en Disco

```
#include "stdafx.h"
#include "AgMp.h"
#include "Guardar.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CGuardar::CGuardar(CWnd* pParent /*=NULL*/):CDialog(CGuardar::IDD, pParent){
//{{AFX_DATA_INIT(CGuardar)
m_nombre = _T("");
//}}AFX_DATA_INIT
}

void CGuardar::DoDataExchange(CDataExchange* pDX){
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CGuardar)
DDX_Control(pDX, IDOK, m_ok);
DDX_Control(pDX, IDCANCEL, m_cancel);
DDX_Text(pDX, IDC_nombre, m_nombre);
DDV_MaxChars(pDX, m_nombre, 8);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CGuardar, CDialog)
//{{AFX_MSG_MAP(CGuardar)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

void CGuardar::OnOK() {
CString auxi;
UpdateData(TRUE);
UpdateData(FALSE);
if (m_nombre!="") {
    auxi=m_nombre;
    auxi+="Agm";
    DeleteFile(auxi);
    CopyFile("AgData.Rst",auxi,1);
    m_nombre+="Agp";
    DeleteFile(m_nombre);
    CopyFile("AgData2.Rst",m_nombre,1);
    DeleteFile("Data1.Agr");
    DeleteFile("Data2.Agr");
    DeleteFile("Data3.Agr");
    DeleteFile("Data4.Agr");
    DeleteFile("Data5.Agr");
    DeleteFile("Data6.Agr");
    CopyFile("Resul1.Agr","Data1.Agr",1);
    CopyFile("Resul2.Agr","Data2.Agr",1);
    CopyFile("Resul3.Agr","Data3.Agr",1);
    CopyFile("Resul4.Agr","Data4.Agr",1);
    CopyFile("Resul6.Agr","Data5.Agr",1);
    CopyFile("Resul7.Agr","Data6.Agr",1);
}
```

```
        CDialog::OnOK();
    } //if
    else MessageBox("Por favor ingrese un nombre","AGMP");
}

int CGuardar::DoModal() {return CDialog::DoModal();}
```


Archivo de encabezado Guardar.h

```
#include "RoundButton.h"
#ifndef AFX_GUARDAR_H__43684C81_53C1_11D5_AB3A_BE1F29FB9A58__INCLUDED_
#define AFX_GUARDAR_H__43684C81_53C1_11D5_AB3A_BE1F29FB9A58__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CGuardar : public CDialog {
// Construction
public:
    CGuardar(CWnd* pParent = NULL); // standard constructor
// Dialog Data
   //{{AFX_DATA(CGuardar)
    enum { IDD = IDD_GUARDAR_DIALOG };
    CRoundButton m_ok;
    CRoundButton m_cancel;
    CString m_nombre;
    }//}}AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CGuardar)
    public:
        virtual int DoModal();
    protected:
        virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }//}}AFX_VIRTUAL
// Implementation
protected:
    // Generated message map functions
   //{{AFX_MSG(CGuardar)
    virtual void OnOK();
    }//}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
#endif AFX_GUARDAR_H__43684C81_53C1_11D5_AB3A_BE1F29FB9A58__INCLUDED_
```

Archivo TBM.cpp

Gráfico de las tasas de natalidad, mortalidad y crecimiento

```
#include "stdafx.h"
#include "AgMp.h"
#include "TBM.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CTBM::CTBM(CWnd* pParent /*=NULL*/): CDialog(CTBM::IDD, pParent){
   //{{AFX_DATA_INIT(CTBM)
        // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT
}

void CTBM::DoDataExchange(CDataExchange* pDX){
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CTBM)
    DDX_Control(pDX, IDOK, m_ok);
    DDX_Control(pDX, IDC_Copiar, m_copiar);
    DDX_Control(pDX, IDC_MSCHART1, m_Chart);
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CTBM, CDialog)
    {{{AFX_MSG_MAP(CTBM)
        ON_WM_SHOWWINDOW()
        ON_BN_CLICKED(IDC_Copiar, OnCopiar)
    }}}AFX_MSG_MAP
END_MESSAGE_MAP()

long CTBM::numdatos(void) {
    long x;
    FILE *qtr;
    qtr=fopen("Confi.Agc","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        fscanf(qtr,"%ld",&x);
        fclose(qtr);
    }
    return x;
}

void CTBM::OnShowWindow(BOOL bShow, UINT nStatus) {
    COleSafeArray saRet;
    FILE *qtr;
    long iter=0;
    long index[2];
    double val;
    char aux[20];
    CDialog::OnShowWindow(bShow, nStatus);
    iter=numdatos();
    DWORD numElements[] = {iter,3};
```

```

saRet.Create(VT_R8, 2, numElements);
qtr=fopen("Resul6.Agr","rt");
for(index[0]=0;index[0]<iter; index[0]++) {
    index[1]=0;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
    index[1]=1;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
    index[1]=2;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
}
fclose(qtr);
m_Chart.SetChartData(saRet.Detach());
for (int i=0;i<iter;i++) {
    m_Chart.SetRow(i+1);
    itoa(i,aux,10);
    m_Chart.SetRowLabel(aux);
}
m_Chart.Refresh();
}

void CTBM::OnCopiar() {m_Chart.EditCopy();}

```

Archivo de encabezado TBM.h

```
#include "RoundButton.h"
//{{AFX_INCLUDES()
#include "mschart.h"
//}}AFX_INCLUDES
#if !defined(AFX_TBM_H__9B4F0DA2_5D6F_11D5_AB3A_C13EF882B249__INCLUDED_)
#define AFX_TBM_H__9B4F0DA2_5D6F_11D5_AB3A_C13EF882B249__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CTBM : public CDialog {
// Construction
public:
    CTBM(CWnd* pParent = NULL); // standard constructor
    long numdatos(void);

// Dialog Data
    {{{AFX_DATA(CTBM)
    enum { IDD = IDD_TBM_DIALOG };
    CRoundButton    m_ok;
    CRoundButton    m_copiar;
    CMSChart        m_Chart;
    }}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    {{{AFX_VIRTUAL(CTBM)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    {{{AFX_MSG(CTBM)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnCopiar();
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
#endif // !defined(AFX_TBM_H__9B4F0DA2_5D6F_11D5_AB3A_C13EF882B249__INCLUDED_)
```

Archivo de encabezado TBM2.h

```
#include "RoundButton.h"
//{{AFX_INCLUDES()
#include "mschart.h"
//}}AFX_INCLUDES
#if !defined(AFX_TBM2_H__5CFB4FA1_5DE7_11D5_AB3A_98F307D0CD40__INCLUDED_)
#define AFX_TBM2_H__5CFB4FA1_5DE7_11D5_AB3A_98F307D0CD40__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CTBM2 : public CDialog{
// Construction
public:
    CTBM2(CWnd* pParent = NULL); // standard constructor
    long numdatos(void);

// Dialog Data
    //{AFX_DATA(CTBM2)
    enum { IDD = IDD_TBM2_DIALOG };
    CRoundButton    m_ok;
    CRoundButton    m_copiar;
    CMSChart        m_Chart;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CTBM2)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(CTBM2)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnCopiar();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif // !defined(AFX_TBM2_H__5CFB4FA1_5DE7_11D5_AB3A_98F307D0CD40__INCLUDED_)
```



```

#include "stdafx.h"
#include "AgMp.h"
#include "TBM2.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CTBM2::CTBM2(CWnd* pParent /*=NULL*/) : CDialog(CTBM2::IDD, pParent){
   //{{AFX_DATA_INIT(CTBM2)
        // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT
}

void CTBM2::DoDataExchange(CDataExchange* pDX){
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CTBM2)
    DDX_Control(pDX, IDOK, m_ok);
    DDX_Control(pDX, IDC_Copiar, m_copiar);
    DDX_Control(pDX, IDC_MSCHART1, m_Chart);
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CTBM2, CDialog)
    {{{AFX_MSG_MAP(CTBM2)
    ON_WM_SHOWWINDOW()
    ON_BN_CLICKED(IDC_Copiar, OnCopiar)
    }}}AFX_MSG_MAP
END_MESSAGE_MAP()

long CTBM2::numdatos(void) {
    long x;
    FILE *qtr;
    qtr=fopen("Confi.Agc","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        fscanf(qtr,"%ld",&x);
        fclose(qtr);
    }//if
    return x;
}

void CTBM2::OnShowWindow(BOOL bShow, UINT nStatus) {
    COleSafeArray saRet;
    FILE *qtr;
    long iter=0;
    long index[2];
    double val;
    char aux[20];
    CDialog::OnShowWindow(bShow, nStatus);
    iter=numdatos();
    DWORD numElements[] = {iter,3};
    saRet.Create(VT_R8, 2, numElements);
    qtr=fopen("Resul7.Agr","rt");
    for(index[0]=0;index[0]<iter; index[0]++) {
        index[1]=0;
        fscanf(qtr,"%lf",&val);
    }
}

```

```

        saRet.PutElement(index, &val);
        index[1]=1;
        fscanf(qtr,"%lf",&val);
        saRet.PutElement(index, &val);
        index[1]=2;
        fscanf(qtr,"%lf",&val);
        saRet.PutElement(index, &val);
    }//for
    fclose(qtr);
    m_Chart.SetChartData(saRet.Detach());
    for (int i=0;i<iter;i++) {
        m_Chart.SetRow(i+1);
        itoa(i,aux,10);
        m_Chart.SetRowLabel(aux);
    }//for
    m_Chart.Refresh();
}

void CTBM2::OnCopiar() {m_Chart.EditCopy();}

```

Archivo Deptiempo.cpp Optimizador de Modelos

```
#include "stdafx.h"
#include "AgMp.h"
#include "Resultados.h"
#include "math.h"
#include "Min_cuadrado.h"
#include "DepTiempo.h"
#include "Mutavan.h"
#include "Crossavan.h"
#include "FileEditDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

int sele=0;
double mayor_tope;

CDepTiempo::CDepTiempo(CWnd* pParent /*=NULL*/) : CDialog(CDepTiempo::IDD, pParent){
    DeleteFile("AgData.Dat");
    DeleteFile("Betas.Dat");
    DeleteFile("Confi.Agc");
    DeleteFile("temporal");
    DeleteFile("tempo");
    DeleteFile("n.dat"); //{{{AFX_DATA_INIT(CDepTiempo)
    m_b0a = 0.0;
    m_b0b = 1.0;
    m_b1a = 0.0;
    m_b1b = 1.0;
    m_iter = 10;
    m_mejor = FALSE;
    m_pcross = 0.9;
    m_pmuta = 0.4;
    m_tambus = 500;
    m_tope = 0.0;
    m_elitismo = FALSE;
    //}}}AFX_DATA_INIT
}

void CDepTiempo::DoDataExchange(CDataExchange* pDX) {
    CDialog::DoDataExchange(pDX);
    //{{{AFX_DATA_MAP(CDepTiempo)
    DDX_Control(pDX, IDC_Mutavan, m_mutavan);
    DDX_Control(pDX, IDC_Crossavan, m_crossavan);
    DDX_Control(pDX, IDC_Abrir, m_abrir);
    DDX_Control(pDX, IDCANCEL, m_salir);
    DDX_Control(pDX, IDC_ejecutar, m_ejecutar);
    DDX_Text(pDX, IDC_B0a, m_b0a);
    DDX_Text(pDX, IDC_B0b, m_b0b);
    DDX_Text(pDX, IDC_B1a, m_b1a);
    DDX_Text(pDX, IDC_B1b, m_b1b);
    DDX_Text(pDX, IDC_iter, m_iter);
    DDX_Check(pDX, IDC_mejor, m_mejor);
    DDX_Text(pDX, IDC_pcross, m_pcross);
    //}}}AFX_DATA_MAP
}
```



```

        DDV_MinMaxDouble(pDX, m_pcross, 0., 1.);
        DDX_Text(pDX, IDC_pmuta, m_pmuta);
        DDV_MinMaxDouble(pDX, m_pmuta, 0., 1.);
        DDX_Text(pDX, IDC_Tambus, m_tambus);
        DDX_Text(pDX, IDC_tope, m_tope);
        DDX_Check(pDX, IDC_elitismo, m_elitismo);
    //}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDepTiempo, CDialog)
    //{{AFX_MSG_MAP(CDepTiempo)
    ON_BN_CLICKED(IDC_ejecutar, OnEjecutar)
    ON_WM_SHOWWINDOW()
    ON_BN_CLICKED(IDC_torneo, Ontorneo)
    ON_BN_CLICKED(IDC_ruleta, Onruleta)
    ON_BN_CLICKED(IDC_Crossavan, OnCrossavan)
    ON_BN_CLICKED(IDC_Mutavan, OnMutavan)
    ON_BN_CLICKED(IDC_Abrir, OnAbrir)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_EVENTSINK_MAP(CDepTiempo, CDialog)
    //{{AFX_EVENTSINK_MAP(CDepTiempo)
    //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()

void CDepTiempo::Ontorneo() {sele=0;}
void CDepTiempo::Onruleta() {sele=1;}

void CDepTiempo::OnShowWindow(BOOL bShow, UINT nStatus) {
    CDialog::OnShowWindow(bShow, nStatus);
    CDialog::CheckRadioButton(IDC_torneo, IDC_ruleta, IDC_torneo);
}

void CDepTiempo::OnEjecutar() {
    CResultados Re;
    FILE *qtr;
    int cros=0, mut=0, extremos=0, menor=0;
    UpdateData(TRUE);
    qtr=fopen("AgData.Dat", "rt");
    if
(m_iter>0&&(m_tope>=mayor_tope||m_tope==0)&&m_b0a<=m_b0b&&m_b1a<=m_b1b&&m_tambus>
=2&&qtr!=NULL) {
        fclose(qtr);
        qtr=fopen("cro.agc", "rt");
        if (qtr!=NULL) {
            fscanf(qtr, "%d", &cros);
            fscanf(qtr, "%d", &extremos);
            fclose(qtr);
        }
        qtr=fopen("mut.agc", "rt");
        if (qtr!=NULL) {
            fscanf(qtr, "%d", &mut);
            fscanf(qtr, "%d", &menor);
            fclose(qtr);
        }
        DeleteFile("confi.agc");
        qtr=fopen("confi.agc", "at");
        fprintf(qtr, "%lf\n", m_b0a);
    }
}

```

```

        fprintf(qtr,"%lf\n",m_b0b);
        fprintf(qtr,"%lf\n",m_b1a);
        fprintf(qtr,"%lf\n",m_b1b);
        fprintf(qtr,"%lf\n",m_tope);
        fprintf(qtr,"%ld\n",m_iter);
        fprintf(qtr,"%d\n",cros);
        fprintf(qtr,"%lf\n",m_pcross);
        fprintf(qtr,"%d\n",extremos);
        fprintf(qtr,"%d\n",mut);
        fprintf(qtr,"%lf\n",m_pmuta);
        fprintf(qtr,"%d\n",menor);
        fprintf(qtr,"%d\n",sele);
        fprintf(qtr,"%d\n",m_mejor);
        fprintf(qtr,"%d\n",m_elitismo);
        fprintf(qtr,"%ld",m_tambus);
        fclose(qtr);
        UpdateData(FALSE);
        Re.DoModal();
    }
}
else MessageBox("Por favor ingrese correctamente los datos","Modelo de crecimiento
Logistico");
}

void CDepTiempo::OnMutavan() {
    CMutavan Mut;
    Mut.DoModal();
}

void CDepTiempo::OnCrossavan() {
    CCrossavan Cro;
    Cro.DoModal();
}

void CDepTiempo::OnAbrir() {
    CFileEditDlg dlg;
    Mincua MC;
    int n = 1;
    FILE *qtr;
    double x,mayor_tope;
    DeleteFile("Confi.Agc");
    qtr=fopen("Confi.Agc","at");
    fprintf(qtr,"%lf",0.0);
    fprintf(qtr,"%lf",0.0);
    fprintf(qtr,"%lf",0.0);
    fprintf(qtr,"%lf",0.0);
    fprintf(qtr,"%lf",0.0);
    fprintf(qtr,"%lf",0.0);
    fclose(qtr);
    dlg.DoModal();
    TRACE ("Files entered :\n");
    POSITION pos = dlg.m_FileEditCtrl.GetStartPosition();
    while (pos) {
        CString x = dlg.m_FileEditCtrl.GetNextPathName(pos);
        TRACE ("%03d - %s\n",n, x);
        n++;
    }
    TRACE ("Directory entered :\n %s\n",dlg.m_String);
    MC.proceso();
    qtr=fopen("AgData.Dat","rt");
    if (qtr!=NULL) {

```

```

UpdateData(TRUE);
m_b0a=floor(MC._a());
m_b0b=ceil(MC._a());
m_b1a=MC._b()-pow(10,-2);
m_b1b=MC._b()+pow(10,-2);
fscanf(qtr,"%lf",&x);
mayor_tope=x;
while (!feof(qtr)) {
    fscanf(qtr,"%lf",&x);
    if (x>mayor_tope) mayor_tope=x;
} //while
fclose(qtr);
m_tope=mayor_tope+1;
UpdateData(FALSE);
} //if
}

```

Archivo de encabezado Deptiempo.h

```
#include "RoundButton.h"
#ifndef AFX_DEPTIEMPO_H__48F28381_2808_11D5_AB39_D1177D6EC063__INCLUDED_
#define AFX_DEPTIEMPO_H__48F28381_2808_11D5_AB39_D1177D6EC063__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CDepTiempo : public CDialog {
// Construction
public:
    CDepTiempo(CWnd* pParent = NULL); // standard constructor

// Dialog Data
   //{{AFX_DATA(CDepTiempo)
    enum { IDD = IDD_DEPTIEMPO_DIALOG };
    CRoundButton    m_mutavan;
    CRoundButton    m_crossavan;
    CRoundButton    m_abrir;
    CRoundButton    m_salir;
    CRoundButton    m_ejecutar;
    double          m_b0a;
    double          m_b0b;
    double          m_b1a;
    double          m_b1b;
    long            m_iter;
    BOOL            m_mejor;
    double          m_pcross;
    double          m_pmuta;
    long            m_tambus;
    double          m_tope;
    BOOL            m_elitismo;
    }//}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CDepTiempo)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }//}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
   //{{AFX_MSG(CDepTiempo)
    afx_msg void Onejecutar();
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void Ontorneo();
    afx_msg void Onruleta();
    afx_msg void OnCrossavan();
    afx_msg void OnMutavan();
    afx_msg void OnAbrir();
    DECLARE_EVENTSINK_MAP()
    }//}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
#endif AFX_DEPTIEMPO_H__48F28381_2808_11D5_AB39_D1177D6EC063__INCLUDED_
```

Archivo de encabezado Ag.hpp **Algoritmo genético de Optimización**

```

#define MAXLON 20
class AG {
private:
    struct persona {
        BOOL cromos1[MAXLON+1];
        BOOL cromos2[MAXLON+1];
        double ada;
        double dis;
        double a,b;
    };
    struct pobla {
        persona individuo;
        pobla *sig;
    }*pob;
    struct datos {
        double dato;
        datos *sig;
    }*datox;

    double Y[3];
    double SCR,R2,SCE;

    long n;//número de datos
    double tope;//valor maximo de la funcion
    long iter;//iteraciones
    long tambus;//tamaño de búsqueda paralela
    long notambus;//valores del intervalo
    double pcross;//prob. crossover
    double pmuta;//prob. mutacion
    double b0a,b0b,b1a,b1b;//intervalos de los parametros
    int tcross,tmuta,sele;
    BOOL extremos,menor,mejor,elitismo;//opciones
    int longitud;//longitud de los cromosomas
    double prome;//aptitud promedio
    double mayor_a;//b0 de la mayor aptitd
    double mayor_b;//b1 de la mayor aptitd
protected:
    void agregar_datos(double);
    void carga_datos(void);
    void sacar(void);//elimina el primero de la lista
    void sacar2(void);
    void ruleta(void);//método de la ruleta
    void torneo(void);//método del torneo
    void crossover1(double,BOOL);//1 punto
    void crossoverU(double);//uniforme
    void mutacion1(double,int);//1 punto
    void mutacionU(double,int);//uniforme
    double genotipo(int []);//base 10
    double adaptacion(double,double);//funcion de adaptacion
    double genotipo1(int A[]) {return genotipo(A)*(double)((b0b-b0a)/notambus)+b0a;}
    //genotipo de b0
    double genotipo2(int A[]) {return genotipo(A)*(double)((b1b-b1a)/notambus)+b1a;}
    //genotipo de b1
    long random(long x) {return (long)floor(x*rand()/RAND_MAX);}
    //número aleatorio uniforme discreto

```



```

1));}
    long preferencia_menor(double w) {return (long)(0.5*(sqrt(1+8*random((long)(w*(w+1)*0.5)))-
//número aleatorio con preferencia en los menores
public:
    void crear(void); //crea la lista
    void destruir(void); //destruye la lista
    void lee_configuracion(void); //lee
    void inicial(void); //genera la pob inicial
    void halla_Sumas(void); //halla las sumas en y
    void ordenar(void); //ordena la lista
    void crossover(void); //crossover
    void mutacion(void); //mutacion
    void seleccion(void); //selecciona
    void reemplazo(void); //reemplaza la poblacion
    void maypro(void); //halla el mayor y el promedio
    double halla_sce(void); //halla la sce;
    void halla_resultados(void);
    BOOL converge(long);
    double _a1(void) {return mayor_a;} //retorna b0
    double _b1(void) {return mayor_b;} //retorna b1
    double _SCE1(void) {return SCE;} //retorna SCE
    long _iter(void) {return iter;} //retorna el numero de iteraciones
    double _SCR(void) {return SCR;}
    double _R2(void) {return R2;}
};

void AG::agregar_datos(double y) {
    datos *auxil;
    if (datox==NULL) {
        datox=(datos *)malloc(sizeof(datos));
        datox->dato=y;
        datox->sig=NULL;
    } //if
    else {
        auxil=datox;
        while(auxil->sig!=NULL) auxil=auxil->sig;
        auxil->sig=(datos *)malloc(sizeof(datos));
        auxil=auxil->sig;
        auxil->sig=NULL;
        auxil->dato=y;
    } //else
}

void AG::carga_datos(void) {
    FILE *qtr;
    datox=NULL;
    double y;
    long t=0;
    qtr=fopen("AgData.Dat","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        while (!feof(qtr)) {
            t++;
            fscanf(qtr,"%lf",&y);
            agregar_datos(y);
        } //while
        fclose(qtr);
    } //if
}

```

```

void AG::lee_configuracion(void) {
FILE *qtr;
    qtr=fopen("confi.agc","rt");
    fscanf(qtr,"%lf",&b0a);
    fscanf(qtr,"%lf",&b0b);
    fscanf(qtr,"%lf",&b1a);
    fscanf(qtr,"%lf",&b1b);
    fscanf(qtr,"%lf",&tope);
    fscanf(qtr,"%ld",&iter);
    fscanf(qtr,"%d",&tcross);
    fscanf(qtr,"%lf",&pcross);
    fscanf(qtr,"%d",&extremos);
    fscanf(qtr,"%d",&tmuta);
    fscanf(qtr,"%lf",&pmuta);
    fscanf(qtr,"%d",&menor);
    fscanf(qtr,"%d",&sele);
    fscanf(qtr,"%d",&mejor);
    fscanf(qtr,"%d",&elitismo);
    fscanf(qtr,"%ld",&tambus);
    fclose(qtr);
    longitud=(int)(log10((double)tambus)/log10(2.0));
    notambus=(long)pow(2,longitud);
    carga_datos();
}

void AG::crear(void) {
    srand((unsigned int)time(NULL));
    pob=NULL;
}

void AG::sacar(void) {
pobla *aux;
    aux=pob;
    pob=pob->sig;
    delete aux;
}

void AG::sacar2(void) {
datos *aux;
    aux=datox;
    datox=datox->sig;
    delete aux;
}

void AG::destruir(void) {
    R2=adaptacion(mayor_a,mayor_b);
    while (pob!=NULL) sacar();
    while (datox!=NULL) sacar2();
}

void AG::ordenar(void) {
pobla *aux1,*aux2;
double temp;
    aux1=pob;
    while (aux1->sig!=NULL) {
        aux2=aux1->sig;
        while (aux2!=NULL) {
            if (aux2->individuo.ada>aux1->individuo.ada) {

```

```

        temp=aux1->individuo.ada;
        aux1->individuo.ada=aux2->individuo.ada;
        aux2->individuo.ada=temp;
    } //if
    aux2=aux2->sig;
} //while
aux1=aux1->sig;
} //while
}

void AG::inicial(void) {
    int i,k;
    pobla *aux;
    pob=(pobla *)malloc(sizeof(pobla));
    pob->sig=NULL;
    do {
        for (i=0;i<longitud;i++) pob->individuo.cromo1[i]=(BOOL)random(2);
        for (i=0;i<longitud;i++) pob->individuo.cromo2[i]=(BOOL)random(2);
        pob->individuo.a=genotipo1(pob->individuo.cromo1);
        pob->individuo.b=genotipo2(pob->individuo.cromo2);
        pob->individuo.ada=adaptacion(pob->individuo.a,pob->individuo.b);
    } while (pob->individuo.ada<0);
    aux=pob;
    for (k=1;k<tambus;k++) {
        aux->sig=(pobla *)malloc(sizeof(pobla));
        aux=aux->sig;
        aux->sig=NULL;
        do {
            for (i=0;i<longitud;i++) aux->individuo.cromo1[i]=(BOOL)random(2);
            for (i=0;i<longitud;i++) aux->individuo.cromo2[i]=(BOOL)random(2);
            aux->individuo.a=genotipo1(aux->individuo.cromo1);
            aux->individuo.b=genotipo2(aux->individuo.cromo2);
            aux->individuo.ada=adaptacion(aux->individuo.a,aux->individuo.b);
        } while (aux->individuo.ada<0);
    } //for
    SCR=Y[2]-n*pow(Y[0]/n,2);
}

double AG::genotipo(int A[]) {
    int j;
    double suma=(double)A[0];
    for (j=1;j<longitud;j++) {
        suma*=2;
        suma+=(double)(A[j]);
    } //for
    return suma;
}

void AG::halla_Sumas(void) {
    FILE *qtr;
    long i=0;
    double w;
    Y[0]=0.0;
    Y[1]=0.0;
    Y[2]=0.0;
    qtr=fopen("AgData.Dat","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        while (!feof(qtr)) {

```



```

        i++;
        fscanf(qtr, "%lf", &w);
        if (tope!=0) {
            Y[0]+=log(1/w-1/tope);
            Y[1]+=log(1/w-1/tope)*i;
            Y[2]+=log(1/w-1/tope)*log(1/w-1/tope);
        }
        else {
            Y[0]+=log(w);
            Y[1]+=log(w)*i;
            Y[2]+=log(w)*log(w);
        }
    }
    while
    fclose(qtr);
    n=i;
}

double AG::adaptacion(double b0, double b1) {
    datos *aux;
    double w, y;
    long t=0;
    SCE=0.0;
    aux=datos;
    while (aux!=NULL) {
        y=aux->dato;
        if (tope!=0) {
            w=1/(1/tope+b0*pow(b1, t+1));
            SCE+=pow(log(1/y-1/tope)-log(1/w-1/tope), 2);
        }
        else {
            w=b0*exp(b1*(t+1));
            SCE+=pow(log(y/w), 2);
        }
        aux=aux->sig;
        t++;
    }
    return 1-SCE/(SCR+SCE);
}

void AG::seleccion(void) {
    if (sele==0) torneo();
    else ruleta();
}

void AG::ruleta(void) {
    double acum=0.0;
    pobla *aux, *aux2;
    double k;
    BOOL cromos3[MAXLON+1];
    BOOL cromos4[MAXLON+1];
    int i;
    aux=pob;
    for (i=0; i<tambus/2; i++) {
        acum+=aux->individuo.ada;
        aux=aux->sig;
    }
    aux=pob;
    for (i=0; i<tambus/2; i++) {

```

```

        aux->individuo.dis=aux->individuo.ada/acum;
        aux=aux->sig;
    }//for
    aux=pob;
    acum=0.0;
    for (i=0;i<tambus/2;i++) {
        aux->individuo.dis+=acum;
        acum=aux->individuo.dis;
        aux=aux->sig;
    }//for
    aux=pob;
    if (elitismo==TRUE) {
        for (i=0;i<tambus/2;i++) aux=aux->sig;
    }//if
    else if (mejor==TRUE) aux=aux->sig;
    while (aux!=NULL) {
        k=(double)rand() / RAND_MAX;
        aux2=pob;
        while (aux2!=NULL) {
            if (k<=aux2->individuo.dis) {
                for (i=0;i<longitud;i++) cromos3[i]=aux2->individuo.cromo1[i];
                for (i=0;i<longitud;i++) cromos4[i]=aux2->individuo.cromo2[i];
                break;
            }//if
            aux2=aux2->sig;
        }//while
        for (i=0;i<longitud;i++) aux->individuo.cromo1[i]=cromos3[i];
        for (i=0;i<longitud;i++) aux->individuo.cromo2[i]=cromos4[i];
        aux=aux->sig;
    }//while
}

void AG::torneo(void) {
    pobla *aux,*aux2;
    long W,i;
    int k;

    aux=pob;
    if (elitismo==TRUE) {
        for (i=0;i<tambus/2;i++) aux=aux->sig;
    }//if
    else if (mejor==TRUE) aux=aux->sig;
    while (aux!=NULL) {
        W=preferencia_menor(tambus/2);
        aux2=pob;
        for (i=0;i<W;i++) aux2=aux2->sig;
        for (k=0;k<longitud;k++) aux->individuo.cromo1[k]=aux2->individuo.cromo1[k];
        for (k=0;k<longitud;k++) aux->individuo.cromo2[k]=aux2->individuo.cromo2[k];
        aux=aux->sig;
    }//while
}

void AG::crossover(void) {
    if (tcross==0||extremos==TRUE) crossover1(pcross,extremos);
    else crossoverU(pcross);
}

void AG::crossover1(double P,BOOL extremos) {
    double U;
    pobla *aux;

```

```

int i,p1,p2;
BOOL cad[MAXLON+1];
aux=pob;
if (elitismo==TRUE) {
    for (i=0;i<tambus/2;i++) aux=aux->sig;
}
else if (mejor==TRUE) {
    for (i=0;i<2;i++) aux=aux->sig;
}
while (aux!=NULL&&aux->sig!=NULL) {
    U=(double)rand()/RAND_MAX;
    if (U<P) {
        if (extremos==TRUE) p1=0;
        else p1=(int)random(longitud);
        for (i=0;i<=p1;i++) cad[i]=aux->sig->individuo.cromo1[i];
        for (i=0;i<=p1;i++) aux->sig->individuo.cromo1[i]=aux->individuo.cromo1[i];
        for (i=0;i<=p1;i++) aux->individuo.cromo1[i]=cad[i];
        if (extremos==TRUE) p2=0;
        else p2=(int)random(longitud);
        for (i=0;i<=p2;i++) cad[i]=aux->sig->individuo.cromo2[i];
        for (i=0;i<=p2;i++) aux->sig->individuo.cromo2[i]=aux->individuo.cromo2[i];
        for (i=0;i<=p2;i++) aux->individuo.cromo2[i]=cad[i];
    }
    aux=aux->sig->sig;
}
}

```

```

void AG::crossoverU(double P) {
double U;
pobla *aux;
int i;
BOOL cad;
aux=pob;
if (elitismo==TRUE) {
    for (i=0;i<tambus/2;i++) aux=aux->sig;
}
else if (mejor==TRUE) aux=aux->sig;
while (aux!=NULL&&aux->sig!=NULL) {
    for (i=0;i<longitud;i++) {
        U=(double)rand()/RAND_MAX;
        if (U<P) {
            cad=aux->individuo.cromo1[i];
            aux->individuo.cromo1[i]=aux->sig->individuo.cromo1[i];
            aux->sig->individuo.cromo1[i]=cad;
        }
        U=(double)rand()/RAND_MAX;
        if (U<P) {
            cad=aux->individuo.cromo2[i];
            aux->individuo.cromo2[i]=aux->sig->individuo.cromo2[i];
            aux->sig->individuo.cromo2[i]=cad;
        }
    }
    aux=aux->sig->sig;
}
}

```

```

void AG::mutacion(void) {
    if (menor==TRUE) {
        reemplazo();
    }
}

```

```

        maypro();
    }//if
    if (tmuta==0) mutacion1(pmuta,menor);
    else mutacionU(pmuta,menor);
}

void AG::mutacion1(double P,BOOL menor) {
double U;
pobla *aux;
int p;

    aux=pob;
    if (elitismo==TRUE) {
        for (p=0;p<tambus/2;p++) aux=aux->sig;
    }//if
    else if (mejor==TRUE) aux=aux->sig;
    while (aux!=NULL) {
        U=(double)rand()/RAND_MAX;
        if (U<P) {
            p=random(longitud);
            if(aux->individuo.ada<prom||menor==FALSE) {
                if (aux->individuo.cromo1[p]==1) aux->individuo.cromo1[p]=0;
                else aux->individuo.cromo1[p]=1;
            }//if
            p=random(longitud);
            if(aux->individuo.ada<prom||menor==FALSE) {
                if (aux->individuo.cromo2[p]==1) aux->individuo.cromo2[p]=0;
                else aux->individuo.cromo2[p]=1;
            }//if
        }//if
        aux=aux->sig;
    }//while
}

void AG::mutacionU(double P,BOOL menor) {
double U;
pobla *aux;
int i;

    aux=pob;
    if (elitismo==TRUE) {
        for (i=0;i<tambus/2;i++) aux=aux->sig;
    }//if
    else if (mejor==TRUE) aux=aux->sig;
    while (aux!=NULL) {
        if(aux->individuo.ada<prom||menor==FALSE) {
            for (i=0;i<longitud;i++) {
                U=(double)rand()/RAND_MAX;
                if (U<P) {
                    if (aux->individuo.cromo1[i]==1) aux->individuo.cromo1[i]=0;
                    else aux->individuo.cromo1[i]=1;
                }//if
                U=(double)rand()/RAND_MAX;
                if (U<P) {
                    if (aux->individuo.cromo2[i]==1) aux->individuo.cromo2[i]=0;
                    else aux->individuo.cromo2[i]=1;
                }//if
            }//for
        }//if
    }
}

```

```

        aux=aux->sig;
    }//while
}

void AG::reemplazo(void) {
    pobla *aux;
    int i;
    aux=pob;
    if (elitismo==TRUE) {
        for (i=0;i<tambus/2;i++) aux=aux->sig;
    }//if
    else if (mejor==TRUE) aux=aux->sig;
    while (aux!=NULL) {
        aux->individuo.a=genotipo1(aux->individuo.cromo1);
        aux->individuo.b=genotipo2(aux->individuo.cromo2);
        aux->individuo.ada=adaptacion(aux->individuo.a,aux->individuo.b);
        aux=aux->sig;
    }//while
}

void AG::maypro(void) {
    pobla *aux;
    aux=pob;
    R2=pob->individuo.ada;
    mayor_a=pob->individuo.a;
    mayor_b=pob->individuo.b;
    prome=0.0;
    while (aux!=NULL) {
        prome+=aux->individuo.ada;
        aux=aux->sig;
    }//while
    prome/=(double)tambus;
}

BOOL AG::converge(long x) {
    pobla *aux;
    long cont=0;
    if (x>=iter) return TRUE;
    aux=pob;
    while (aux->sig!=NULL) {
        if (aux->individuo.a==mayor_a) {
            if (aux->individuo.b==mayor_b) cont++;
        }//if
        aux=aux->sig;
    }//while
    return ((BOOL)cont/tambus>=0.95);
}

```


Archivo de encabezado Min_cuadrado.h

Método de los mínimos cuadrados

```

class Mincua {
private:
    double SCE,SCR,R2;
    long n;
    double Y[3],X[2][2],B[2];
    double tope;
protected:
    long halla_xty(void);
    void halla_inv_xtx(void);
    void halla_betas(void);
    void halla_sce(void);
public:
    void proceso(void);
    double _a(void) {return exp(B[0]);}
    double _b(void) {
        if (tope!=0) return exp(B[1]);
        return B[1];
    }
    long _gl(void) {return n-2;}
    double _SCE(void) {return SCE;}
    double _SCR(void) {return SCR;}
    double _R2(void) {return R2;}
};

void Mincua::proceso(void) {
    FILE *qtr;
    qtr=fopen("Confi.Agc","rt");
    fscanf(qtr,"%lf",&tope);
    fscanf(qtr,"%lf",&tope);
    fscanf(qtr,"%lf",&tope);
    fscanf(qtr,"%lf",&tope);
    fscanf(qtr,"%lf",&tope);
    fscanf(qtr,"%lf",&tope);
    fclose(qtr);
    n=halla_xty();
    halla_inv_xtx();
    halla_betas();
    halla_sce();
}

long Mincua::halla_xty(void) {
    FILE *qtr;
    double w;
    long cont=0;
    qtr=fopen("AgData.Dat","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        Y[0]=0.0;
        Y[1]=0.0;
        Y[2]=0.0;
        while (!feof(qtr)) {
            cont++;
            fscanf(qtr,"%lf",&w);
            if (tope!=0) {
                Y[0]+=(double)log(1/w-1/tope);
                Y[1]+=(double)(log(1/w-1/tope)*cont);
            }
        }
    }
}

```

```

        }
        else {
            Y[0]+=(double)log(w);
            Y[1]+=(double)(log(w)*cont);
            Y[2]+=(double)(log(w)*log(w));
        }
    }
    fclose(qtr);
}

//if
return cont;
}

void Mincua::halla_inv_xtx(void) {
    int i,j;
    X[0][0]=(double)(n*(n+1)*(2*n+1)/6);
    X[0][1]=(double)(-n*(n+1)/2);
    X[1][1]=(double)n;
    X[1][0]=X[0][1];
    for (i=0;i<2;i++) {
        for (j=0;j<2;j++) X[i][j]/=((double)((n*n)*(n+1)*(2*n+1)/6)-(double)((n*n)*(n+1)*(n+1)/4));
    }
}

void Mincua::halla_betas(void) {
    int i,j;
    for (i=0;i<2;i++) {
        B[i]=0.0;
        for (j=0;j<2;j++) B[i]+=X[i][j]*Y[j];
    }
}

void Mincua::halla_sce(void) {
    SCE=Y[2]-(B[0]*Y[0]+B[1]*Y[1]);
    SCR=Y[2]-n*pow(Y[0]/n,2);
    R2=1-SCE/(SCE+SCR);
}

```

Archivo Resultados.cpp

Resultados de la Optimización

```
#include "stdafx.h"
#include "AgMp.h"
#include "Resultados.h"
#include "math.h"
#include "Min_cuadrado.h"
#include "Grafico.h"
#include "Graerror.h"
#include "intervalos.h"
#include "Ag.hpp"
#include "ProgressWnd.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

AG G3;

CResultados::CResultados(CWnd* pParent /*=NULL*/)
: CDialog(CResultados::IDD, pParent) {
//{{AFX_DATA_INIT(CResultados)
m_b0 = 0.0;
m_b0m = 0.0;
m_b1 = 0.0;
m_b1m = 0.0;
m_Fempm = 0.0;
m_modelo = _T("");
m_modelom = _T("");
m_r2 = 0.0;
m_r2m = 0.0;
m_sce = 0.0;
m_scem = 0.0;
m_mce = 0.0;
m_mcem = 0.0;
m_Femp = 0.0;
//}}AFX_DATA_INIT
}

void CResultados::DoDataExchange(CDataExchange* pDX) {
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CResultados)
DDX_Control(pDX, IDOK, m_ok);
DDX_Control(pDX, IDC_Gra, m_gra);
DDX_Control(pDX, IDC_Graerror, m_graerror);
DDX_Control(pDX, IDC_Intervalos, m_inter);
DDX_Text(pDX, IDC_b0, m_b0);
DDX_Text(pDX, IDC_b0M, m_b0m);
DDX_Text(pDX, IDC_b1, m_b1);
DDX_Text(pDX, IDC_b1M, m_b1m);
DDX_Text(pDX, IDC_FempM, m_Fempm);
DDX_Text(pDX, IDC_modelo, m_modelo);
DDX_Text(pDX, IDC_modeloM, m_modelom);
DDX_Text(pDX, IDC_R2, m_r2);
DDX_Text(pDX, IDC_R2M, m_r2m);
}}
```



```

        DDX_Text(pDX, IDC_sce, m_sce);
        DDX_Text(pDX, IDC_sceM, m_sceM);
        DDX_Text(pDX, IDC_mce, m_mce);
        DDX_Text(pDX, IDC_mceM, m_mceM);
        DDX_Text(pDX, IDC_Femp, m_Femp);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CResultados, CDialog)
    //{{AFX_MSG_MAP(CResultados)
    ON_WM_SHOWWINDOW()
    ON_BN_CLICKED(IDC_Gra, OnGra)
    ON_BN_CLICKED(IDC_Graerror, OnGraerror)
    ON_BN_CLICKED(IDC_Intervalos, OnIntervalos)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

void CResultados::Proceso(void) {
    long contador=0;
    G3.crear();
    G3.lee_configuracion();
    G3.halla_Sumas();
    G3.inicial();
    G3.ordenar();
    G3.maypro();
    CProgressWnd wndProgress(this, "Procesando el Algoritmo Genético", TRUE);
    wndProgress.SetRange(0, G3.iter());
    wndProgress.SetText("Por favor, Espere un momento...");
    while (!G3.converge(contador)) {
        if (wndProgress.Cancelled()) break;
        G3.seleccion();
        G3.reemplazo(); G3.ordenar(); G3.maypro();
        G3.crossover();
        G3.reemplazo(); G3.ordenar(); G3.maypro();
        G3.mutacion();
        G3.reemplazo(); G3.ordenar(); G3.maypro();
        contador++;
        wndProgress.StepIt();
        wndProgress.PeekAndPump();
        if (wndProgress.Cancelled()) break;
    } //while
    G3.destruir();
}

```

```

void CResultados::OnShowWindow(BOOL bShow, UINT nStatus) {
    char aux[20];
    FILE *ptr;
    double tope;
    double gl;
    Mincua MC1;
    CDialog::OnShowWindow(bShow, nStatus);
    MC1.proceso();
    UpdateData(TRUE);
    m_b0=MC1._a();
    m_b1=MC1._b();
    gl=MC1._gl();
    m_sce=MC1._SCE();
    m_mce=m_sce/gl;
    m_r2=MC1._R2();
}

```

```

if (m_r2<0.0) m_r2=0.0;
if (m_mce!=0) m_Femp=MC1._SCR()/m_mce-gl;
else m_Femp=0.0;
if (m_Femp<0.0) m_Femp=0.0;
Proceso();
m_b0m=G3._a1();
m_b1m=G3._b1();
m_scem=G3._SCE1();
m_mcem=m_scem/gl;
m_r2m=G3._R2();
if (m_r2m<0.0) m_r2m=0.0;
if (m_mcem!=0) m_Fempm=G3._SCR()/m_mcem-gl;
else m_Fempm=0.0;
if (m_Fempm<0) m_Fempm=0.0;
ptr=fopen("Confi.Agc","rt");
fscanf(ptr,"%lf",&tope);
fscanf(ptr,"%lf",&tope);
fscanf(ptr,"%lf",&tope);
fscanf(ptr,"%lf",&tope);
fscanf(ptr,"%lf",&tope);
fclose(ptr);
DeleteFile("Resul.Agr");
ptr=fopen("Resul.Agr","at");
fprintf(ptr,"%lf\n",m_b0m);
fprintf(ptr,"%lf\n",m_b1m);
fprintf(ptr,"%lf\n",m_b0);
fprintf(ptr,"%lf\n",m_b1);
fprintf(ptr,"%lf\n",m_mce);
fprintf(ptr,"%lf",m_mcem);
fclose(ptr);
if (tope!=0) {
    m_modelo="P(t)=1/(";
    gcvt(1/tope,4,aux);
    m_modelo+=aux;
    m_modelo+="*";
    gcvt(m_b0,4,aux);
    m_modelo+=aux;
    m_modelo+="*";
    gcvt(m_b1,4,aux);
    m_modelo+=aux;
    m_modelo+="^t)";
    m_modelom="P(t)=1/(";
    gcvt(1/tope,4,aux);
    m_modelom+=aux;
    m_modelom+="*";
    gcvt(m_b0m,4,aux);
    m_modelom+=aux;
    m_modelom+="*";
    gcvt(m_b1m,4,aux);
    m_modelom+=aux;
    m_modelom+="^t)";
}
}
else {
    m_modelo="P(t)=";
    gcvt(m_b0,4,aux);
    m_modelo+=aux;
    m_modelo+="*exp(";
    gcvt(m_b1,4,aux);
    m_modelo+=aux;

```

```

        m_modelo+="*t";
        m_modelom="P(t)=";
        gcvt(m_b0m,4,aux);
        m_modelom+=aux;
        m_modelom+="*exp(";
        gcvt(m_b1m,4,aux);
        m_modelom+=aux;
        m_modelom+="*t)";
    }//else
    UpdateData(FALSE);
}

void CResultados::OnGra() {
    CGrafico Gr;
    Gr.DoModal();
}

void CResultados::OnGraerror() {
    CGraerror Gr;
    Gr.DoModal();
}

void CResultados::OnIntervalos() {
    Cintervalos I;
    I.DoModal();
}

void CResultados::OnOK() {CDialog::OnCancel();}

```

Archivo de encabezado Resultados.h

```
#include "RoundButton.h"
#ifndef AFX_RESULTADOS_H_DF6EC181_0D9A_11D5_AB39_B96C9A074A52__INCLUDED_
#define AFX_RESULTADOS_H_DF6EC181_0D9A_11D5_AB39_B96C9A074A52__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
class CResultados : public CDialog{
// Construction
public:
    CResultados(CWnd* pParent = NULL); // standard constructor
    void Proceso(void);

// Dialog Data
   //{{AFX_DATA(CResultados)
    enum { IDD = IDD_RESULTADOS_DIALOG };
    CRoundButton    m_ok;
    CRoundButton    m_gra;
    CRoundButton    m_graerror;
    CRoundButton    m_inter;
    double          m_b0;
    double          m_b0m;
    double          m_b1;
    double          m_b1m;
    double          m_Fempr;
    CString         m_modelo;
    CString         m_modelom;
    double          m_r2;
    double          m_r2m;
    double          m_sce;
    double          m_scem;
    double          m_mce;
    double          m_mcem;
    double          m_Femp;
    }//}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CResultados)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }//}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(CResultados)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnGra();
    afx_msg void OnGraerror();
    afx_msg void OnIntervalos();
    virtual void OnOK();
    }//}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
#endif AFX_RESULTADOS_H_DF6EC181_0D9A_11D5_AB39_B96C9A074A52__INCLUDED_
```

Archivo intervalos.cpp

Intervalos de confianza para los betas

```
#include "stdafx.h"
#include "AgMp.h"
#include "intervalos.h"
#include "math.h"
#include "TablaT.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

TablaT T;
double C=0.025;
long numero;
double B0,B1,B0m,B1m,mce,mcem;

Cintervalos::Cintervalos(CWnd* pParent /*=NULL*/)
: CDialog(Cintervalos::IDD, pParent){
//{{AFX_DATA_INIT(Cintervalos)
m_b0inf = 0.0;
m_b0inf2 = 0.0;
m_b0sup = 0.0;
m_b0sup2 = 0.0;
m_b1inf = 0.0;
m_b1inf2 = 0.0;
m_b1sup = 0.0;
m_b1sup2 = 0.0;
m_b0t2 = 0.0;
m_b1t = 0.0;
m_b1t2 = 0.0;
m_b0t = 0.0;
m_t = 0.0;
//}}AFX_DATA_INIT
}

void Cintervalos::DoDataExchange(CDataExchange* pDX){
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(Cintervalos)
DDX_Control(pDX, IDOK, m_ok);
DDX_Text(pDX, IDC_B0inf, m_b0inf);
DDX_Text(pDX, IDC_B0inf2, m_b0inf2);
DDX_Text(pDX, IDC_B0sup, m_b0sup);
DDX_Text(pDX, IDC_B0sup2, m_b0sup2);
DDX_Text(pDX, IDC_B1inf, m_b1inf);
DDX_Text(pDX, IDC_B1inf2, m_b1inf2);
DDX_Text(pDX, IDC_B1sup, m_b1sup);
DDX_Text(pDX, IDC_B1sup2, m_b1sup2);
DDX_Text(pDX, IDC_b0t2, m_b0t2);
DDX_Text(pDX, IDC_b1t, m_b1t);
DDX_Text(pDX, IDC_b1t2, m_b1t2);
DDX_Text(pDX, IDC_bot, m_b0t);
DDX_Text(pDX, IDC_t, m_t);
//}}AFX_DATA_MAP
}
```



```

BEGIN_MESSAGE_MAP(Cintervalos, CDialog)
//{{AFX_MSG_MAP(Cintervalos)
    ON_BN_CLICKED(IDC_005, On005)
    ON_BN_CLICKED(IDC_010, On010)
    ON_BN_CLICKED(IDC_025, On025)
    ON_BN_CLICKED(IDC_050, On050)
    ON_BN_CLICKED(IDC_100, On100)
    ON_WM_SHOWWINDOW()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

long Cintervalos::numdatos(char ss[]) {
long contador=0;
double x;
FILE *qtr;
    qtr=fopen("AgData.Dat","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        while (!feof(qtr)) {
            fscanf(qtr,"%lf",&x);
            contador++;
        }
    }
    fclose(qtr);
    return contador;
}

void Cintervalos::actualiza(double m_t) {
    m_b0inf=B0-m_t*sqrt((double)mcem*X[0]);
    m_b0sup=B0+m_t*sqrt((double)mcem*X[0]);
    m_b0inf2=B0-m_t*sqrt((double)mce*X[0]);
    m_b0sup2=B0+m_t*sqrt((double)mce*X[0]);
    m_b0t=B0/sqrt(mcem*X[0]);
    m_b0t2=B0/sqrt(mce*X[0]);
    m_b1inf=B1-m_t*sqrt((double)mcem*X[1]);
    m_b1sup=B1+m_t*sqrt((double)mcem*X[1]);
    m_b1inf2=B1-m_t*sqrt((double)mce*X[1]);
    m_b1sup2=B1+m_t*sqrt((double)mce*X[1]);
    m_b1t=B1/sqrt(mcem*X[1]);
    m_b1t2=B1/sqrt(mce*X[1]);
}

void Cintervalos::On005() {
    UpdateData(TRUE);
    m_t=T.valor_tabla(0.005,numero-2);
    actualiza(m_t);
    UpdateData(FALSE);
}

void Cintervalos::On010() {
    UpdateData(TRUE);
    m_t=T.valor_tabla(0.010,numero-2);
    actualiza(m_t);
    UpdateData(FALSE);
}

void Cintervalos::On025() {
    UpdateData(TRUE);

```

```

        m_t=T.valor_tabla(0.025,numero-2);
        actualiza(m_t);
        UpdateData(FALSE);
    }

void Cintervalos::On050() {
    UpdateData(TRUE);
    m_t=T.valor_tabla(0.05,numero-2);
    actualiza(m_t);
    UpdateData(FALSE);
}

void Cintervalos::On100() {
    UpdateData(TRUE);
    m_t=T.valor_tabla(0.10,numero-2);
    actualiza(m_t);
    UpdateData(FALSE);
}

void Cintervalos::halla_xtx(long n) {
    int i;
    X[0]=(n/6)*(n+1)*(2*n+1);
    X[1]=n;
    for (i=0;i<2;i++) X[i]/=((n*n/6)*(n+1)*(2*n+1)-(n*n/4)*(n+1)*(n+1));
}

void Cintervalos::OnShowWindow(BOOL bShow, UINT nStatus) {
    FILE *qtr;
    CDialog::OnShowWindow(bShow, nStatus);
    CheckRadioButton(IDC_005,IDC_100,IDC_025);
    numero=numdatos("AgData.Dat");
    halla_xtx(numero);
    qtr=fopen("Resul.Agr","rt");
    if (qtr!=NULL) {
        fscanf(qtr,"%lf",&B0);
        fscanf(qtr,"%lf",&B1);
        fscanf(qtr,"%lf",&B0m);
        fscanf(qtr,"%lf",&B1m);
        fscanf(qtr,"%lf",&mce);
        fscanf(qtr,"%lf",&mcem);
        fclose(qtr);
    }
    UpdateData(TRUE);
    m_t=T.valor_tabla(C,numero-2);
    actualiza(m_t);
    UpdateData(FALSE);
}

```

Archivo de encabezado intervalos.h

```
#include "RoundButton.h"
#if !defined(AFX_INTERVALOS_H__AEC2C6C2_186D_11D5_AB39_813D7B8EFE62__INCLUDED_)
#define AFX_INTERVALOS_H__AEC2C6C2_186D_11D5_AB39_813D7B8EFE62__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class Cintervalos : public CDialog {
private:
    double X[2];
public:
    Cintervalos(CWnd* pParent = NULL); // standard constructor
    long numdatos(char []);
    void halla_xtx(long);
    void actualiza(double);
// Dialog Data
   //{{AFX_DATA(Cintervalos)
    enum { IDD = IDD_INTERVALOS_DIALOG };
    CRoundButton m_ok;
    double m_b0inf;
    double m_b0inf2;
    double m_b0sup;
    double m_b0sup2;
    double m_b1inf;
    double m_b1inf2;
    double m_b1sup;
    double m_b1sup2;
    double m_b0t2;
    double m_b1t;
    double m_b1t2;
    double m_b0t;
    double m_t;
    //}}AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(Cintervalos)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
protected:
   //{{AFX_MSG(Cintervalos)
    afx_msg void On005();
    afx_msg void On010();
    afx_msg void On025();
    afx_msg void On05();
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void On10();
    afx_msg void On050();
    afx_msg void On100();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
#endif //
!defined(AFX_INTERVALOS_H__AEC2C6C2_186D_11D5_AB39_813D7B8EFE62__INCLUDED_)
```


Archivo Grafico.cpp

Gráfico de la tendencia del modelo optimizado

```
#include "stdafx.h"
#include "AgMp.h"
#include "math.h"
#include "Grafico.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CGrafico::CGrafico(CWnd* pParent /*=NULL*/)
: CDialog(CGrafico::IDD, pParent){
//{{AFX_DATA_INIT(CGrafico)
// NOTE: the ClassWizard will add member initialization here
//}}AFX_DATA_INIT
}

void CGrafico::DoDataExchange(CDataExchange* pDX){
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CGrafico)
DDX_Control(pDX, IDC_Copiar, m_copiar);
DDX_Control(pDX, IDOK, m_ok);
DDX_Control(pDX, IDC_MSCHART1, m_Chart);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CGrafico, CDialog)
//{{AFX_MSG_MAP(CGrafico)
ON_WM_SHOWWINDOW()
ON_BN_CLICKED(IDC_Copiar, OnCopiar)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

long CGrafico::numdatos(char ss[]) {
long contador=0;
double x;
FILE *qtr;
qtr=fopen("AgData.Dat","rt");
if (qtr!=NULL) {
rewind(qtr);
while (!feof(qtr)) {
fscanf(qtr,"%lf",&x);
contador++;
} //while
} //if
fclose(qtr);
return contador;
}

void CGrafico::OnShowWindow(BOOL bShow, UINT nStatus) {
COleSafeArray saRet;
FILE *qtr;
long iter=0;
```

```

long index[2];
double val,tope;
double B0a,B1a,B0b,B1b;
char aux[20];
CDialog::OnShowWindow(bShow, nStatus);
qtr=fopen("Resul.Agr","rt");
fscanf(qtr,"%lf",&B0a);
fscanf(qtr,"%lf",&B1a);
fscanf(qtr,"%lf",&B0b);
fscanf(qtr,"%lf",&B1b);
fclose(qtr);
qtr=fopen("Confi.Agc","rt");
fscanf(qtr,"%lf",&tope);
fscanf(qtr,"%lf",&tope);
fscanf(qtr,"%lf",&tope);
fscanf(qtr,"%lf",&tope);
fscanf(qtr,"%lf",&tope);
fclose(qtr);
iter=numdatos("AgData.Dat");
DWORD numElements[] = {iter,3};
saRet.Create(VT_R8, 2, numElements);
qtr=fopen("AgData.Dat","rt");
for(index[0]=0;index[0]<iter; index[0]++) {
    index[1]=0;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index, &val);
    index[1]=1;
    if (tope!=0) val= 1/( 1/tope+B0a*pow(B1a,(index[0])+1));
    else val=B0a*exp(B1a*(index[0]+1));
    saRet.PutElement(index, &val);
    index[1]=2;
    if (tope!=0) val= 1/( 1/tope+B0b*pow(B1b,(index[0])+1));
    else val=B0b*exp(B1b*(index[0]+1));
    saRet.PutElement(index, &val);
}
//for
fclose(qtr);
m_Chart.SetChartData(saRet.Detach());
for (int i=0;i<iter;i++) {
    m_Chart.SetRow(i+1);
    itoa(i,aux,10);
    m_Chart.SetRowLabel(aux);
}
//for
m_Chart.Refresh();
}
void CGrafico::OnCopiar() {m_Chart.EditCopy();}

```

Archivo de encabezado Grafico.h

```
#include "RoundButton.h"
//{{AFX_INCLUDES()
#include "mschart.h"
//}}AFX_INCLUDES
#ifndef AFX_GRAFICO_H_D12FAF42_13B1_11D5_AB39_8BA52D8B6D5D__INCLUDED_
#define AFX_GRAFICO_H_D12FAF42_13B1_11D5_AB39_8BA52D8B6D5D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CGrafico : public CDialog{
// Construction
public:
    CGrafico(CWnd* pParent = NULL); // standard constructor
    long numdatos(char ss[]);

// Dialog Data
    {{{AFX_DATA(CGrafico)
    enum { IDD = IDD_GRAFICO_DIALOG };
    CRoundButton    m_copiar;
    CRoundButton    m_ok;
    CMSChart        m_Chart;
    }}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    {{{AFX_VIRTUAL(CGrafico)
    protected:
        virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    {{{AFX_MSG(CGrafico)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnCopiar();
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
#ifndef AFX_GRAFICO_H_D12FAF42_13B1_11D5_AB39_8BA52D8B6D5D__INCLUDED_
```

Archivo Graerror.cpp

Gráfico de los residuos del modelo optimizado

```
#include "stdafx.h"
#include "AgMp.h"
#include "Graerror.h"
#include "math.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CGraerror::CGraerror(CWnd* pParent /*=NULL*/)
: CDialog(CGraerror::IDD, pParent) {
   //{{AFX_DATA_INIT(CGraerror)
    // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT
}

void CGraerror::DoDataExchange(CDataExchange* pDX){
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CGraerror)
    DDX_Control(pDX, IDOK, m_ok);
    DDX_Control(pDX, IDC_Copiar, m_copiar);
    DDX_Control(pDX, IDC_MSCHART1, m_Chart);
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CGraerror, CDialog)
    {{{AFX_MSG_MAP(CGraerror)
    ON_WM_SHOWWINDOW()
    ON_BN_CLICKED(IDC_Copiar, OnCopiar)
    }}}AFX_MSG_MAP
END_MESSAGE_MAP()

long CGraerror::numdatos(char ss[]) {
    long contador=0;
    double x;
    FILE *qtr;
    qtr=fopen("AgData.Dat","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        while (!feof(qtr)) {
            fscanf(qtr,"%lf",&x);
            contador++;
        }
    }
    fclose(qtr);
    return contador;
}

void CGraerror::OnShowWindow(BOOL bShow, UINT nStatus) {
    COleSafeArray saRet;
    FILE *qtr;
    long iter=0;
    long index[2];
```

```

double val,w,tope;
double B0a,B1a,B0b,B1b;
char aux[20];
CDialog::OnShowWindow(bShow, nStatus);
qtr=fopen("Resul.Agr","rt");
fscanf(qtr,"%lf",&B0a);
fscanf(qtr,"%lf",&B1a);
fscanf(qtr,"%lf",&B0b);
fscanf(qtr,"%lf",&B1b);
fclose(qtr);
qtr=fopen("Confi.Agc","rt");
fscanf(qtr,"%lf",&tope);
fscanf(qtr,"%lf",&tope);
fscanf(qtr,"%lf",&tope);
fscanf(qtr,"%lf",&tope);
fscanf(qtr,"%lf",&tope);
fscanf(qtr,"%lf",&tope);
fclose(qtr);
iter=numdatos("AgData.Dat");
DWORD numElements[] = {iter,2};
saRet.Create(VT_R8, 2, numElements);
qtr=fopen("AgData.Dat","rt");
for(index[0]=0;index[0]<iter;index[0]++) {
    index[1]=1;
    fscanf(qtr,"%lf",&val);
    if (tope!=0) w=1/(1/tope+B0a*pow(B1a,index[0]+1));
    else w=B0a*exp(B1a*(index[0]+1));
    w=val-w;
    saRet.PutElement(index,&w);
    index[1]=0;
    if (tope!=0) w=1/(1/tope+B0b*pow(B1b,index[0]+1));
    else w=B0b*exp(B1b*(index[0]+1));
    w=val-w;
    saRet.PutElement(index,&w);
}
fclose(qtr);
m_Chart.SetChartData(saRet.Detach());
for (int i=0;i<iter;i++) {
    m_Chart.SetRow(i+1);
    itoa(i,aux,10);
    m_Chart.SetRowLabel(aux);
}
m_Chart.Refresh();
}

```

```

void CGraerror::OnCopiar() {m_Chart.EditCopy();}

```


Archivo de encabezado Graerror.h

```
#include "RoundButton.h"
//{{AFX_INCLUDES()}
#include "mschart.h"
//}}AFX_INCLUDES
#if !defined(AFX_GRAERROR_H_AEC2C6C1_186D_11D5_AB39_813D7B8EFE62__INCLUDED_)
#define AFX_GRAERROR_H_AEC2C6C1_186D_11D5_AB39_813D7B8EFE62__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CGraerror : public CDialog{
// Construction
public:
    CGraerror(CWnd* pParent = NULL); // standard constructor
    long numdatos(char []);
// Dialog Data
    {{{AFX_DATA(CGraerror)
    enum { IDD = IDD_GRAERROR_DIALOG };
    CRoundButton    m_ok;
    CRoundButton    m_copiar;
    CMSChart        m_Chart;
    }}}AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
    {{{AFX_VIRTUAL(CGraerror)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }}}AFX_VIRTUAL
// Implementation
protected:
    {{{AFX_MSG(CGraerror)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnCopiar();
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
!defined(AFX_GRAERROR_H_AEC2C6C1_186D_11D5_AB39_813D7B8EFE62__INCLUDED_)
```

Archivo Depotras.cpp

Seleccionador del mejor modelo

```
#include "stdafx.h"
#include "AgMp.h"
#include "DepOtras.h"
#include "Resultados2.h"
#include "Mutavan.h"
#include "Crossavan.h"
#include "FileEditDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

int sele2=0;
double mayor_tope2;

DepOtras::DepOtras(CWnd* pParent /*=NULL*/) : CDialog(DepOtras::IDD, pParent) {
    DeleteFile("AgData.Dat");
    DeleteFile("Betas.Dat");
    DeleteFile("Confi.Agc");
    DeleteFile("temporal");
    DeleteFile("tempo");
    DeleteFile("n.dat");
   //{{AFX_DATA_INIT(DepOtras)
    m_elitismo = FALSE;
    m_iter = 10;
    m_mejor = FALSE;
    m_pcross = 0.9;
    m_pmuta = 0.4;
    m_tambus = 8;
    m_tope = 0.0;
   //}}AFX_DATA_INIT
}

void DepOtras::DoDataExchange(CDataExchange* pDX) {
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(DepOtras)
    DDX_Control(pDX, IDC_Mutavan, m_mutavan);
    DDX_Control(pDX, IDC_Crossavan, m_crossavan);
    DDX_Control(pDX, IDC_Abrir, m_abrir);
    DDX_Control(pDX, IDCANCEL, m_salir);
    DDX_Control(pDX, IDC_ejecutar, m_ejecutar);
    DDX_Check(pDX, IDC_elitismo, m_elitismo);
    DDX_Text(pDX, IDC_iter, m_iter);
    DDX_Check(pDX, IDC_mejor, m_mejor);
    DDX_Text(pDX, IDC_pcross, m_pcross);
    DDV_MinMaxDouble(pDX, m_pcross, 0., 1.);
    DDX_Text(pDX, IDC_pmuta, m_pmuta);
    DDV_MinMaxDouble(pDX, m_pmuta, 0., 1.);
    DDX_Text(pDX, IDC_Tambus, m_tambus);
    DDX_Text(pDX, IDC_tope, m_tope);
   //}}AFX_DATA_MAP
}
```

```

BEGIN_MESSAGE_MAP(DepOtras, CDialog)
   //{{AFX_MSG_MAP(DepOtras)
    ON_BN_CLICKED(IDC_ejecutar, Onejecutar)
    ON_BN_CLICKED(IDC_torneo, Ontorneo)
    ON_BN_CLICKED(IDC_ruleta, Onruleta)
    ON_WM_SHOWWINDOW()
    ON_BN_CLICKED(IDC_Crossavan, OnCrossavan)
    ON_BN_CLICKED(IDC_Mutavan, OnMutavan)
    ON_BN_CLICKED(IDC_Abrir, OnAbrir)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_EVENTSINK_MAP(DepOtras, CDialog)
   //{{AFX_EVENTSINK_MAP(DepOtras)
   //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()

void DepOtras::Ontorneo() {sele2=0;}
void DepOtras::Onruleta() {sele2=1;}

void DepOtras::OnShowWindow(BOOL bShow, UINT nStatus) {
    CDialog::OnShowWindow(bShow, nStatus);
    CDialog::CheckRadioButton(IDC_torneo, IDC_ruleta, IDC_torneo);
}

void DepOtras::Onejecutar() {
    CResultados2 Re;
    FILE *qtr;
    int cros=0, mut=0, extremos=0, menor=0;
    UpdateData(TRUE);
    qtr=fopen("AgData.Dat", "rt");
    if (m_iter>0&&(m_tope>=mayor_tope2||m_tope==0)&&m_tambus>=2&&qtr!=NULL) {
        fclose(qtr);
        qtr=fopen("cro.agc", "rt");
        if (qtr!=NULL) {
            fscanf(qtr, "%d", &cros);
            fscanf(qtr, "%d", &extremos);
            fclose(qtr);
        }
        qtr=fopen("mut.agc", "rt");
        if (qtr!=NULL) {
            fscanf(qtr, "%d", &mut);
            fscanf(qtr, "%d", &menor);
            fclose(qtr);
        }
        DeleteFile("confi.agc");
        qtr=fopen("confi.agc", "at");
        fprintf(qtr, "%lf\n", 0);
        fprintf(qtr, "%ld\n", m_iter);
        fprintf(qtr, "%d\n", cros);
        fprintf(qtr, "%lf\n", m_pcross);
        fprintf(qtr, "%d\n", extremos);
        fprintf(qtr, "%d\n", mut);
        fprintf(qtr, "%lf\n", m_pmuta);
        fprintf(qtr, "%d\n", menor);
        fprintf(qtr, "%d\n", sele2);
        fprintf(qtr, "%d\n", m_mejor);
        fprintf(qtr, "%d\n", m_elitismo);
        fprintf(qtr, "%ld", m_tambus);
    }
}

```



```

        fclose(qtr);
        qtr=fopen("confi.agd","at");
        fprintf(qtr,"%lf\n",0);
        fprintf(qtr,"%d\n",1);
        fprintf(qtr,"%d\n",1);
        fprintf(qtr,"%d\n",1);
        fprintf(qtr,"%d\n",1);
        fprintf(qtr,"%d\n",1);
        fclose(qtr);
        UpdateData(FALSE);
        Re.DoModal();
    }//if
    else MessageBox("Por favor ingrese correctamente los datos","Modelo de crecimiento
Logistico");
}

void DepOtras::OnMutavan() {
    CMutavan Mut;
    Mut.DoModal();
}

void DepOtras::OnCrossavan() {
    CCrossavan Cro;
    Cro.DoModal();
}

void DepOtras::OnAbrir() {
    CFileEditDlg dlg;
    int n = 1;
    FILE *qtr;
    double x,mayor_tope;
    dlg.DoModal();
    TRACE ("nFiles entered :n");
    POSITION pos = dlg.m_FileEditCtrl.GetStartPosition();
    while (pos) {
        CString x = dlg.m_FileEditCtrl.GetNextPathName(pos);
        TRACE ("%03d - %s\n",n, x);
        n++;
    }//while
    TRACE ("Directory entered :n %s\n\n",dlg.m_String);
    qtr=fopen("AgData.Dat","rt");
    if (qtr!=NULL) {
        fscanf(qtr,"%lf",&x);
        UpdateData(TRUE);
        mayor_tope=x;
        while (!feof(qtr)) {
            fscanf(qtr,"%lf",&x);
            fscanf(qtr,"%lf",&x);
            fscanf(qtr,"%lf",&x);
            fscanf(qtr,"%lf",&x);
            fscanf(qtr,"%lf",&x);
            if (x>mayor_tope) mayor_tope=x;
        }//while
        fclose(qtr);
        m_tope=0;
        UpdateData(FALSE);
    }//if
}

```

Archivo de encabezado Depotras.h

```
#include "RoundButton.h"
#ifndef AFX_DEPOTRAS_H__AADE0213_2818_11D5_AB39_D1177D6EC063__INCLUDED_
#define AFX_DEPOTRAS_H__AADE0213_2818_11D5_AB39_D1177D6EC063__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class DepOtras : public CDialog{
// Construction
public:
    DepOtras(CWnd* pParent = NULL); // standard constructor
// Dialog Data
    {{{AFX_DATA(DepOtras)
    enum { IDD = IDD_DEPOTRAS_DIALOG };
    CRoundButton    m_mutavan;
    CRoundButton    m_crossavan;
    CRoundButton    m_abrir;
    CRoundButton    m_salir;
    CRoundButton    m_ejecutar;
    BOOL            m_elitismo;
    long            m_iter;
    BOOL            m_mejor;
    double          m_pcross;
    double          m_pmuta;
    long            m_tambus;
    double          m_tope;
    }}}AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
    {{{AFX_VIRTUAL(DepOtras)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }}}AFX_VIRTUAL
// Implementation
protected:
    // Generated message map functions
    {{{AFX_MSG(DepOtras)
    afx_msg void Onejecutar();
    afx_msg void Ontorneo();
    afx_msg void Onruleta();
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnCrossavan();
    afx_msg void OnMutavan();
    afx_msg void OnAbrir();
    DECLARE_EVENTSINK_MAP()
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

{{{AFX_INSERT_LOCATION}}
#endif //
#endif(AFX_DEPOTRAS_H__AADE0213_2818_11D5_AB39_D1177D6EC063__INCLUDED_)
```

Archivo Ag2.h

Algoritmo genético de Selección

```
#define MAXLON 5
```

```
class AG2 {
```

```
private:
```

```
    struct persona {
```

```
        BOOL cromos[MAXLON];
```

```
        double ada;
```

```
        double dis;
```

```
    };
```

```
    struct pobla {
```

```
        persona individuo;
```

```
        pobla *sig;
```

```
    }*pob2;
```

```
    double SCE,MCE,R2,Femp;
```

```
    long n;//número de datos
```

```
    double tope;//valor maximo de la funcion
```

```
    long iter;//iteraciones
```

```
    long tambus;//tamaño de búsqueda paralela
```

```
    double pcross;//prob. crossover
```

```
    double pmuta;//prob. mutacion
```

```
    int tcross,tmuta,sele;
```

```
    BOOL extremos,menor,mejor,elitismo;//opciones
```

```
    int longitud;//longitud de los cromosomas
```

```
    double mayor;//mayor aptitud
```

```
    BOOL mayor_cro[MAXLON];
```

```
    double prome;//aptitud promedio
```

```
protected:
```

```
    void sacar(void);//elimina el primero de la lista
```

```
    void ruleta(void);//método de la ruleta
```

```
    void torneo(void);//método del torneo
```

```
    void crossover1(double,BOOL);//1 punto
```

```
    void crossoverU(double);//uniforme
```

```
    void mutacion1(double,int);//1 punto
```

```
    void mutacionU(double,int);//uniforme
```

```
    double adaptacion(BOOL A[]);//funcion de adaptacion
```

```
    long random(long x) {return (long)floor(x*rand()/RAND_MAX);}
```

```
    //número aleatorio uniforme discreto
```

```
    long preferencia_menor(double w) {return (long)(0.5*(sqrt(1+8*random((long)(w*(w+1)*0.5)))-
```

```
1));}
```

```
    //número aleatorio con preferencia en los menores
```

```
public:
```

```
    void crear(void);//crea la lista
```

```
    void destruir(void);//destruye la lista
```

```
    void lee_configuracion(void);//lee
```

```
    void inicial(void);//genera la pob inicial
```

```
    void halla_Sumas(void);
```

```
    void ordenar(void);
```

```
    void crossover(void);
```

```
    void mutacion(void);
```

```
    void seleccion(void);
```

```
    void reemplazo(void);
```

```
    void maypro(void);
```

```
    double halla_sce(void);
```

```

        void halla_resultados(void);
        BOOL converge(long);
        double _SCE1(void) {return mayor;}
        double _MCE(void) {return MCE;}
        double _R2(void) {return R2;}
        double _Femp(void) {return Femp;}
        long _Iter(void) {return iter;}
};

void AG2::lee_configuracion(void) {
FILE *qtr;
    qtr=fopen("confi.agc","rt");
    fscanf(qtr,"%lf",&tope);
    fscanf(qtr,"%ld",&iter);
    fscanf(qtr,"%d",&tcross);
    fscanf(qtr,"%lf",&pcross);
    fscanf(qtr,"%d",&extremos);
    fscanf(qtr,"%d",&tmuta);
    fscanf(qtr,"%lf",&pmuta);
    fscanf(qtr,"%d",&menor);
    fscanf(qtr,"%d",&sele);
    fscanf(qtr,"%d",&mejor);
    fscanf(qtr,"%d",&elitismo);
    fscanf(qtr,"%ld",&tambus);
    fclose(qtr);
    longitud=5;
}

void AG2::crear(void) {
    srand((unsigned int)time(NULL));
    pob2=NULL;
}

void AG2::sacar(void) {
pobla *aux;
    aux=pob2;
    pob2=pob2->sig;
    delete aux;
}

void AG2::destruir(void) {while (pob2!=NULL) sacar();}

void AG2::ordenar(void) {
pobla *aux1,*aux2;
double temp;
    aux1=pob2;
    while (aux1->sig!=NULL) {
        aux2=aux1->sig;
        while (aux2!=NULL) {
            if (aux2->individuo.ada>aux1->individuo.ada) {
                temp=aux1->individuo.ada;
                aux1->individuo.ada=aux2->individuo.ada;
                aux2->individuo.ada=temp;
            }
            aux2=aux2->sig;
        }
        aux1=aux1->sig;
    }
}

```



```

void AG2::inicial(void) {
    int i,k;
    pobla *aux;
    pob2=(pobla *)malloc(sizeof(pobla));
    pob2->sig=NULL;
    do {
        for (i=0;i<longitud;i++) pob2->individuo.cromo[i]=(BOOL)random(2);
        pob2->individuo.ada=adaptacion(pob2->individuo.cromo);
    } while (pob2->individuo.ada<0);
    aux=pob2;
    for (k=1;k<tambus;k++) {
        aux->sig=(pobla *)malloc(sizeof(pobla));
        aux=aux->sig;
        aux->sig=NULL;
        do {
            for (i=0;i<longitud;i++) aux->individuo.cromo[i]=(BOOL)random(2);
            aux->individuo.ada=adaptacion(aux->individuo.cromo);
        } while (aux->individuo.ada<0);
    } //for
}

double AG2::adaptacion(BOOL A[]) {
    mincua2 Mc3;
    FILE *qtr;
    DeleteFile("Confi.Agc");
    qtr=fopen("Confi.Agc","at");
    fprintf(qtr,"%if\n",tope);
    fprintf(qtr,"%d\n",A[0]);
    fprintf(qtr,"%d\n",A[1]);
    fprintf(qtr,"%d\n",A[2]);
    fprintf(qtr,"%d\n",A[3]);
    fprintf(qtr,"%d\n",A[4]);
    fclose(qtr);
    Mc3.proceso();
    return Mc3._R2();
}

void AG2::seleccion(void) {
    if (sele==0) torneo();
    else ruleta();
}

void AG2::ruleta(void) {
    double acum=0.0;
    pobla *aux,*aux2;
    double k;
    int cromos2[MAXLON];
    int i;
    aux=pob2;
    for (i=0;i<tambus/2;i++) {
        acum+=aux->individuo.ada;
        aux=aux->sig;
    } //for
    aux=pob2;
    for (i=0;i<tambus/2;i++) {
        aux->individuo.dis=aux->individuo.ada/acum;
        aux=aux->sig;
    } //for
}

```

```

    aux=pob2;
    acum=0.0;
    for (i=0;i<tambus/2;i++) {
        aux->individuo.dis+=acum;
        acum=aux->individuo.dis;
        aux=aux->sig;
    }
    aux=pob2;
    if (elitismo==TRUE) {
        for (i=0;i<tambus/2;i++) aux=aux->sig;
    }
    else if (mejor==TRUE) aux=aux->sig;
    while (aux!=NULL) {
        k=(double)rand()/RAND_MAX;
        aux2=pob2;
        while (aux2!=NULL) {
            if (k<=aux2->individuo.dis) {
                for (i=0;i<longitud;i++) cromos2[i]=aux2->individuo.cromo[i];
                break;
            }
            aux2=aux2->sig;
        }
        for (i=0;i<longitud;i++) aux->individuo.cromo[i]=cromos2[i];
        aux=aux->sig;
    }
}

void AG2::torneo(void) {
    pobla *aux,*aux2;
    long W,i;
    int k;

    aux=pob2;
    if (elitismo==TRUE) {
        for (i=0;i<tambus/2;i++) aux=aux->sig;
    }
    else if (mejor==TRUE) aux=aux->sig;
    while (aux!=NULL) {
        W=preferencia_menor(tambus/2);
        aux2=pob2;
        for (i=0;i<W;i++) aux2=aux2->sig;
        for (k=0;k<longitud;k++) aux->individuo.cromo[k]=aux2->individuo.cromo[k];
        aux=aux->sig;
    }
}

void AG2::crossover(void) {
    if (tcross==0||extremos==TRUE) crossover1(pcross,extremos);
    else crossoverU(pcross);
}

void AG2::crossover1(double P,BOOL extremos) {
    double U;
    pobla *aux;
    int i,p1;
    int cad[MAXLON];
    aux=pob2;
    if (elitismo==TRUE) {
        for (i=0;i<tambus/2;i++) aux=aux->sig;
    }
}

```

```

else if (mejor==TRUE) {
    for (i=0;i<2;i++) aux=aux->sig;
} //if
while (aux!=NULL&&aux->sig!=NULL) {
    U=(double)rand()/RAND_MAX;
    if (U<P) {
        if (extremos==TRUE) p1=0;
        else p1=(int)random(longitud);
        for (i=0;i<=p1;i++) cad[i]=aux->sig->individuo.cromo[i];
        for (i=0;i<=p1;i++) aux->sig->individuo.cromo[i]=aux->individuo.cromo[i];
        for (i=0;i<=p1;i++) aux->individuo.cromo[i]=cad[i];
    } //if
    aux=aux->sig->sig;
} //while
}

void AG2::crossoverU(double P) {
    double U;
    pobla *aux;
    int i;
    int cad;

    aux=pob2;
    if (elitismo==TRUE) {
        for (i=0;i<tambus/2;i++) aux=aux->sig;
    } //if
    else if (mejor==TRUE) aux=aux->sig;
    while (aux!=NULL&&aux->sig!=NULL) {
        for (i=0;i<longitud;i++) {
            U=(double)rand()/RAND_MAX;
            if (U<P) {
                cad=aux->individuo.cromo[i];
                aux->individuo.cromo[i]=aux->sig->individuo.cromo[i];
                aux->sig->individuo.cromo[i]=cad;
            } //if
        } //for
        aux=aux->sig->sig;
    } //while
}

void AG2::mutacion(void) {
    if (menor==TRUE) {
        reemplazo();
        maypro();
    } //if
    if (tmuta==0) mutacion1(pmuta,menor);
    else mutacionU(pmuta,menor);
}

void AG2::mutacion1(double P,BOOL menor) {
    double U;
    pobla *aux;
    int p;

    aux=pob2;
    if (elitismo==TRUE) {
        for (p=0;p<tambus/2;p++) aux=aux->sig;
    } //if
    else if (mejor==TRUE) aux=aux->sig;
    while (aux!=NULL) {
        U=(double)rand()/RAND_MAX;

```

```

        if (U<P) {
            p=random(longitud);
            if(aux->individuo.ada<prom||menor==FALSE) {
                if (aux->individuo.cromo[p]==1) aux->individuo.cromo[p]=0;
                else aux->individuo.cromo[p]=1;
            }
        }
        aux=aux->sig;
    }

void AG2::mutacionU(double P,BOOL menor) {
    double U;
    pobla *aux;
    int i;
    aux=pob2;
    if (elitismo==TRUE) {
        for (i=0;i<tambus/2;i++) aux=aux->sig;
    }
    else if (mejor==TRUE) aux=aux->sig;
    while (aux!=NULL) {
        if(aux->individuo.ada<prom||menor==FALSE) {
            for (i=0;i<longitud;i++) {
                U=(double)rand()/RAND_MAX;
                if (U<P) {
                    if (aux->individuo.cromo[i]==1) aux->individuo.cromo[i]=0;
                    else aux->individuo.cromo[i]=1;
                }
            }
        }
        aux=aux->sig;
    }

void AG2::reemplazo(void) {
    pobla *aux;
    int i;
    aux=pob2;
    if (elitismo==TRUE) {
        for (i=0;i<tambus/2;i++) aux=aux->sig;
    }
    else if (mejor==TRUE) aux=aux->sig;
    while (aux!=NULL) {
        aux->individuo.ada=adaptacion(aux->individuo.cromo);
        aux=aux->sig;
    }

void AG2::maypro(void) {
    pobla *aux;
    aux=pob2;
    mayor=pob2->individuo.ada;
    for (int i=0;i<longitud;i++) mayor_cro[i]=pob2->individuo.cromo[i];
    prome=0.0;
    while (aux!=NULL) {
        prome+=aux->individuo.ada;
        aux=aux->sig;
    }
    prome/= (double)tambus;

```



```

}

BOOL AG2::converge(long x) {
pobla *aux;
long cont=0;
    if (x>=iter) return TRUE;
    aux=pob2;
    while (aux->sig!=NULL) {
        if (aux->individuo.ada==mayor) cont++;
        aux=aux->sig;
    }//while
    return ((BOOL)cont/tambus>=0.95);
}

void AG2::halla_resultados(void) {
FILE *qtr;
mincua2 MC4;
int i;
    DeleteFile("Confi.Agc");
    qtr=fopen("Confi.Agc","at");
    fprintf(qtr,"%lf\n",tope);
    for (i=0;i<5;i++) fprintf(qtr,"%d\n",mayor_cro[i]);
    fclose(qtr);
    MC4.proceso();
    mayor=MC4._sce();
    MCE=MC4._mce();
    R2=MC4._R2();
    Femp=MC4._Femp();
}

```

Archivo Min_cuadrado2.h

Mínimos cuadrados para el modelo de varias variables

```
#define maxdim 200

class mincua2{
private:
    int n,p;
    double Conf,SCE,MCE,SCT,MCT,SCR,MCR,R2,F_empirico;
    double Beta[10];
    double X1[maxdim][5],X[maxdim][5],Y[maxdim];
    double XTX[5][5],Inv[5][5],IXT[5][maxdim];
    double U[5][5],V[5][5],UV[5][5];
    double tope;
    BOOL v0,v1,v2,v3,v4;
protected:
    void tabla(void);
    void intervalos(void);
    void t_student(void);
    int lee_valor_simple(char [],int);
    void carga_datos(void);
    void halla_XTX(void);
    double traza(double [5][5],int);
    double det(double [5][5],int);
    void halla_inversa(void);
    void halla_pseudo(void);
    void multiplica_UV(void);
    void multiplica_UVUT(void);
    void INITIALIZE(double [5][5],int);
    double RMS(double [5][5],int);
    void ZEROOUT(double [5][5],double [5][5],int,int,int);
    void halla_IXT(void);
    void halla_Betas(void);
    void guarda(void);
    void llama_archivo(void);
    void lee_confir(void);
    void crea_dime(void);
public:
    void proceso(void);
    double _sce(void) {return SCE;}
    double _mce(void) {return MCE;}
    double _Femp(void) {return F_empirico;}
    double _R2(void) {return R2;}
    double _tope(void) {return tope;}
};

void mincua2::proceso(void) {
double w;
    lee_confir();
    llama_archivo();
    crea_dime();
    halla_XTX();
    w=fabs(det(XTX,p));
    if (w>pow(10,-5)) halla_inversa();
    else halla_pseudo();
    halla_IXT();
    halla_Betas();
    tabla();
}
```

```

        guarda();
    }

void mincua2::crea_dime(void) {
    FILE *qtr;
    DeleteFile("n.dat");
    qtr=fopen("n.dat","wt");
    fprintf(qtr,"%d %d",n,p);
    fclose(qtr);
}

void mincua2::guarda(void) {
    FILE *qtr;
    int i;
    DeleteFile("Betas.dat");
    qtr=fopen("Betas.dat","at");
    fprintf(qtr,"%lf\n",exp(Beta[0]));
    if (tope==0) {
        for (i=1;i<p;i++) fprintf(qtr,"%lf\n",Beta[i]);
    }
    //if
    else {
        for (i=1;i<p;i++) fprintf(qtr,"%lf\n",exp(Beta[i]));
    }
    //else
    fclose(qtr);
}

int mincua2::lee_valor_simple(char cad[],int i) {
    FILE *qtr;
    int n,j;
    qtr=fopen(cad,"rt");
    for (j=0;j<i;j++) fscanf(qtr,"%d",&n);
    fclose(qtr);
    return n;
}

void mincua2::lee_confi(void) {
    FILE *qtr;
    qtr=fopen("Confi.Agd","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        fscanf(qtr,"%lf",&tope);
        fscanf(qtr,"%d",&v0);
        fscanf(qtr,"%d",&v1);
        fscanf(qtr,"%d",&v2);
        fscanf(qtr,"%d",&v3);
        fscanf(qtr,"%d",&v4);
        fclose(qtr);
    }
    //if
}

void mincua2::llama_archivo(void) {
    FILE *qtr;
    n=0,p=1;
    int i;
    qtr=fopen("AgData.Dat","rt");
    rewind(qtr);
    while (!feof(qtr)) {
        fscanf(qtr,"%lf",&Y[n]);
        X1[n][0]=1;
    }
}

```

```

        fscanf(qtr,"%lf",&X1[n][1]);
        fscanf(qtr,"%lf",&X1[n][2]);
        fscanf(qtr,"%lf",&X1[n][3]);
        fscanf(qtr,"%lf",&X1[n][4]);
        n++;
    }/while
    n--;
    fclose(qtr);
    for (i=0;i<n;i++) X[i][0]=X1[i][0];
    if (v1==TRUE) {
        for (i=0;i<n;i++) X[i][p]=X1[i][1];
        p++;
    }/if
    if (v2==TRUE) {
        for (i=0;i<n;i++) X[i][p]=X1[i][2];
        p++;
    }/if
    if (v3==TRUE) {
        for (i=0;i<n;i++) X[i][p]=X1[i][3];
        p++;
    }/if
    if (v4==TRUE) {
        for (i=0;i<n;i++) X[i][p]=X1[i][4];
        p++;
    }/if
    if (tope==0) {
        for (i=0;i<n;i++) Y[i]=log(Y[i]);
    }/if
    else {
        for (i=0;i<n;i++) Y[i]=log(1/Y[i]-1/tope);
    }/else
}

void mincua2::halla_XTX(void) {
    int i,j,k;
    for (i=0;i<p;i++) {
        for (j=0;j<p;j++) {
            XTX[i][j]=0;
            for (k=0;k<n;k++) XTX[i][j]+=X[k][i]*X[k][j];
        }/for
    }/for
}

double mincua2::det(double N[5][5],int n) {
    int i,j,k=n;
    double A=1,aux;
    double M[10][10];
    for (i=0;i<n;i++) {
        for (j=0;j<n;j++) M[i][j]=N[i][j];
    }/for
    while (k>1) {
        aux=M[0][0];
        A*=aux;
        for (i=0;i<k;i++) M[0][i]/=(double)aux;
        for (i=1;i<k;i++) {
            for (j=1;j<k;j++) M[i][j]=(M[i][0]*M[0][j]);
        }/for
        for (i=1;i<k;i++) {
            for (j=1;j<k;j++) M[i-1][j-1]=M[i][j];
        }
    }
}

```

```

    }//for
    k--;
} //while
return (double)A*(M[0][0]);
}

```

```

void mincua2::halla_IXT(void) {
int i,j,k;
    for (i=0;i<p;i++) {
        for (j=0;j<n;j++) {
            IXT[i][j]=0;
            for (k=0;k<p;k++) IXT[i][j]+=Inv[i][k]*X[j][k];
        } //for
    } //for
}

```

```

void mincua2::halla_Betas(void) {
int i,k;
    for (i=0;i<p;i++) {
        Beta[i]=0;
        for (k=0;k<n;k++) Beta[i]+=IXT[i][k]*Y[k];
    } //for
}

```

```

void mincua2::halla_inversa(void){
int i,j,k=0;
double A=0;
double TMP[10][10];
    for (i=0;i<p;i++) {
        for (j=0;j<p;j++) {
            Inv[i][j]=0;
            TMP[i][j]=XTX[i][j];
        } //for
    } //for
    for (i=0;i<p;i++) Inv[i][i]=1;
    for (k=0;k<p;k++) {
        A=TMP[k][k];
        for (j=0;j<p;j++) {
            TMP[k][j]/=A;
            Inv[k][j]/=A;
        } //for
        if (k<p-1) {
            for (i=k+1;i<p;i++) {
                for (j=0;j<p;j++) Inv[i][j] -= (TMP[i][k]*Inv[k][j]);
            } //for
            for (i=k+1;i<p;i++) {
                for (j=k+1;j<p;j++) TMP[i][j] -= (TMP[i][k]*TMP[k][j]);
            } //for
            for (i=k+1;i<p;i++) TMP[i][k]=0;
        } //if
        if (k>0) {
            for (i=k-1;i>=0;i--) {
                for (j=0;j<p;j++) Inv[i][j] -= (TMP[i][k]*Inv[k][j]);
            } //for
            for (i=k-1;i>=0;i--) {
                for (j=k+1;j<p;j++) TMP[i][j] -= (TMP[i][k]*TMP[k][j]);
            } //for
            for (i=k-1;i>=0;i--) TMP[i][k]=0;
        } //if
    }
}

```

```

        }
    }

void mincua2::INITIALIZE(double A[5][5],int N){
    int J, K;
        for (J=0;J<N;J++) {
            for (K=0;K<N;K++) {
                if (J!=K) A[J][K]=0.0;
                else A[J][K]=1.0;
            }
        }
}

double mincua2::RMS(double A[5][5], int N) {
    double Sum = 0;
    int J;
        for (J=0;J<N;J++) Sum+=A[J][J]*A[J][J];
    return (sqrt(Sum/(double)(N)));
}

void mincua2::ZEROOUT(double A[5][5],double U[5][5],int N,int P,int Q) {
    double Theta;
    double T, C, S;
    int J;
    double XP[10],YP[10];
    double XQ[10],YQ[10];
        Theta=(A[Q][Q]-A[P][P])/(2.0*A[P][Q]);
        T=(fabs(Theta)+sqrt(Theta*Theta+1.0));
        if (Theta>=0) T=1.0/T;
        else T=-1.0/T;
        C=1.0/sqrt(T*T+1.0);
        S=C*T;
        for (J = 0; J < N; J++) {
            XP[J]=A[J][P];
            XQ[J]=A[J][Q];
        }
        A[P][Q]=0;
        A[Q][P]=0;
        A[P][P]=C*C*XP[P]+S*S*XQ[Q]-2.0*C*S*XQ[P];
        A[Q][Q]=S*S*XP[P]+C*C*XQ[Q]+2.0*C*S*XQ[P];
        for (J=0;J<N;J++) {
            if ((J!=P)&&(J!=Q)) {
                A[J][P]=C*XP[J]-S*XQ[J];
                A[P][J]=A[J][P];
                A[J][Q]=C*XQ[J]+S*XP[J];
                A[Q][J]=A[J][Q];
            }
        }
        YP[J]=U[J][P];
        YQ[J]=U[J][Q];
        for (J=0;J<N;J++) {
            U[J][P]=C*YP[J]-S*YQ[J];
            U[J][Q]=S*YP[J]+C*YQ[J];
        }
}

void mincua2::multiplica_UV(void) {

```



```

int i,j,k;
for (i=0;i<p;i++) {
    if (V[i][i]!=0) V[i][i]=(1/V[i][i]);
}
for (i=0;i<p;i++) {
    for (j=0;j<p;j++) {
        UV[i][j]=0;
        for (k=0;k<p;k++) UV[i][j]=U[i][k]*V[k][j];
    }
}
}

```

```

void mincua2::multiplica_UVUT(void) {
int i,j,k;
for (i=0;i<p;i++) {
    for (j=0;j<p;j++) {
        Inv[i][j]=0;
        for (k=0;k<p;k++) Inv[i][j]=UV[i][k]*U[k][j];
    }
}
}

```

```

void mincua2::halla_pseudo(void){
double Epsilon=1E-07;
double MaxSweep=50;
int CountS,State;
double T;
int i,j;
    CountS=0;
    INITIALIZE(U,p);
    for(i=0;i<p;i++) {
        for(j=0;j<p;j++) V[i][j]=XTX[i][j];
    }
do {
    T=RMS(V,p);
    CountS++;
    State=1;
    for(i=0;i<p-1;i++) {
        for(j=i+1;j<p;j++) {
            if ((fabs(V[i][j])/T)>Epsilon) {
                ZEROOUT(V,U,p,i,j);
                State=0;
            }
        }
    }
} while((State!=1)||((MaxSweep>CountS)));
    multiplica_UV();
    multiplica_UVUT();
}

```

```

void mincua2::tabla(void){
int i,j;
double Ylin=0.0,Ylin2=0.0,Ylin3=0.0;
double WW[10];
for (i=0;i<n;i++) Ylin+=Y[i]*Y[i];
for (i=0;i<p;i++) {
    WW[i]=0;
    for (j=0;j<n;j++) WW[i]=X[j][i]*Y[j];
}
for (i=0;i<p;i++) Ylin2+=Beta[i]*WW[i];
SCE=Ylin-Ylin2;
MCE=SCE/(n-p);
}

```

```

for (i=0;i<n;i++) Ylin3+=Y[i];
Ylin3=(double)Ylin3/n;
SCR=Ylin-n*Ylin3*Ylin3;
MCR=SCR/(p-1);
SCT=SCE+SCR;
MCT=SCT/(n-1);
R2=1-SCE/SCT;
if (MCE!=0) F_empirico=MCR/MCE;
else F_empirico=0;

```

```

}

```


Archivo Heuris.h

Búsqueda heurística del mejor modelo

```
#define MAXLON 5

class Heuris {
private:
    double Aptitud;
    BOOL mejor_cromo[MAXLON];
    double mejor;
    double SCE,MCE,R2,Femp;
    double tope;
protected:
    double aptitud(BOOL,BOOL,BOOL,BOOL,BOOL);
    void calcula(void);
public:
    void proceso(void);
    double _SCE1(void) {return mejor;}
    double _MCE(void) {return MCE;}
    double _R2(void) {return R2;}
    double _Femp(void) {return Femp;}
};

void Heuris::proceso(void) {
    FILE *qtr;
    BOOL v0,v1,v2,v3,v4;
    qtr=fopen("Confi.Agc","rt");
    fscanf(qtr,"%f",&tope);
    fclose(qtr);
    mejor=0;
    for (v0=1;v0<=1;v0++) {
        for (v1=0;v1<=1;v1++) {
            for (v2=0;v2<=1;v2++) {
                for (v3=0;v3<=1;v3++) {
                    for (v4=0;v4<=1;v4++) {
                        Aptitud=aptitud(v0,v1,v2,v3,v4);
                        if (Aptitud>mejor) {
                            mejor=Aptitud;
                            mejor_cromo[0]=v0;
                            mejor_cromo[1]=v1;
                            mejor_cromo[2]=v2;
                            mejor_cromo[3]=v3;
                            mejor_cromo[4]=v4;
                        }
                    }
                }
            }
        }
    } //for
    calcula();
}

double Heuris::aptitud(BOOL v0,BOOL v1,BOOL v2,BOOL v3,BOOL v4) {
    mincua2 Mc3;
    FILE *qtr;
    DeleteFile("Confi.Agd");
    qtr=fopen("Confi.Agd","at");
    fprintf(qtr,"%f\n",tope);
    fprintf(qtr,"%d\n",v0);
    fprintf(qtr,"%d\n",v1);
    fprintf(qtr,"%d\n",v2);
    fprintf(qtr,"%d\n",v3);
}
```

```

        fprintf(qtr,"%d\n",v4);
        fclose(qtr);
        Mc3.proceso();
        return Mc3._R2();
    }

void Heuris::calcula(void) {
    FILE *qtr;
    mincua2 MC4;
    int i;

    DeleteFile("Confi.Agd");
    qtr=fopen("Confi.Agd","at");
    fprintf(qtr,"%lf\n",tope);
    for (i=0;i<5;i++) fprintf(qtr,"%d\n",mejor_cromo[i]);
    fclose(qtr);
    MC4.proceso();
    mejor=MC4._sce();
    MCE=MC4._mce();
    R2=MC4._R2();
    Femp=MC4._Femp();
}

```

Archivo Resultados2.cpp

Resultados del Seleccionador

```
#include "stdafx.h"
#include "AgMp.h"
#include "Resultados2.h"
#include "math.h"
#include "ProgressWnd.h"
#include "Min_cuadrado2.h"
#include "Ag2.h"
#include "heuristi.h"
#include "Graerror2.h"
#include "Grafico2.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

mincua2 Mc2;
AG2 G2;
Heuris H;

CResultados2::CResultados2(CWnd* pParent /*=NULL*/): CDialog(CResultados2::IDD, pParent){
//{{AFX_DATA_INIT(CResultados2)
m_b0m = 0.0;
m_b0 = 0.0;
m_b1m = 0.0;
m_b1 = 0.0;
m_b2m = 0.0;
m_b2 = 0.0;
m_b3m = 0.0;
m_b3 = 0.0;
m_b4m = 0.0;
m_b4 = 0.0;
m_Fempm = 0.0;
m_Femp = 0.0;
m_mcem = 0.0;
m_mce = 0.0;
m_r2m = 0.0;
m_r2 = 0.0;
m_scem = 0.0;
m_sce = 0.0;
//}}AFX_DATA_INIT
}

void CResultados2::DoDataExchange(CDataExchange* pDX){
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CResultados2)
DDX_Control(pDX, IDOK, m_ok);
DDX_Control(pDX, IDC_Tamano, m_tama);
DDX_Control(pDX, IDC_Graerror, m_graerro);
DDX_Text(pDX, IDC_b0M, m_b0m);
DDX_Text(pDX, IDC_b0M2, m_b0);
DDX_Text(pDX, IDC_b1M, m_b1m);
DDX_Text(pDX, IDC_b1M2, m_b1);
DDX_Text(pDX, IDC_b2M, m_b2m);
```

```

        DDX_Text(pDX, IDC_b2M2, m_b2);
        DDX_Text(pDX, IDC_b3M, m_b3m);
        DDX_Text(pDX, IDC_b3M2, m_b3);
        DDX_Text(pDX, IDC_b4M, m_b4m);
        DDX_Text(pDX, IDC_b4M2, m_b4);
        DDX_Text(pDX, IDC_FempM, m_Fempm);
        DDX_Text(pDX, IDC_FempM2, m_Femp);
        DDX_Text(pDX, IDC_mceM, m_mcem);
        DDX_Text(pDX, IDC_mceM2, m_mce);
        DDX_Text(pDX, IDC_R2M, m_r2m);
        DDX_Text(pDX, IDC_R2M2, m_r2);
        DDX_Text(pDX, IDC_sceM, m_scem);
        DDX_Text(pDX, IDC_sceM2, m_sce);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CResultados2, CDialog)
    //{{AFX_MSG_MAP(CResultados2)
    ON_WM_SHOWWINDOW()
    ON_BN_CLICKED(IDC_Graerror, OnGraerror)
    ON_BN_CLICKED(IDC_Tamano, OnTamano)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

void CResultados2::Proceso(void) {
    long contador=0;
    G2.crear();
    G2.lee_configuracion();
    G2.inicial();
    G2.ordenar();
    G2.maypro();
    CProgressWnd wndProgress(this, "Procesando el Algoritmo Genético", TRUE);
    wndProgress.SetRange(0, G2.iter());
    wndProgress.SetText("Por favor, Espere un momento...");
    while (!G2.converge(contador)) {
        if (wndProgress.Cancelled()) break;
        G2.seleccion();
        G2.reemplazo(); G2.ordenar(); G2.maypro();
        G2.crossover();
        G2.reemplazo(); G2.ordenar(); G2.maypro();
        G2.mutacion();
        G2.reemplazo(); G2.ordenar(); G2.maypro();
        contador++;
        wndProgress.StepIt();
        wndProgress.PeekAndPump();
        if (wndProgress.Cancelled()) break;
    } //while
    G2.halla_resultados();
    G2.destruir();
}

void CResultados2::OnShowWindow(BOOL bShow, UINT nStatus) {
    FILE *qtr;
    BOOL v0,v1,v2,v3,v4;
    double tope;
    CDialog::OnShowWindow(bShow, nStatus);
    UpdateData(TRUE);
    H.proceso();
}

```

```

qtr=fopen("Confi.Agd","rt");
fscanf(qtr,"%lf",&tope);
fscanf(qtr,"%d",&v0);
fscanf(qtr,"%d",&v1);
fscanf(qtr,"%d",&v2);
fscanf(qtr,"%d",&v3);
fscanf(qtr,"%d",&v4);
fclose(qtr);
qtr=fopen("Betas.dat","rt");
if (qtr!=NULL) {
    if (v0==TRUE) fscanf(qtr,"%lf",&m_b0);
    if (v1==TRUE) fscanf(qtr,"%lf",&m_b1);
    if (v2==TRUE) fscanf(qtr,"%lf",&m_b2);
    if (v3==TRUE) fscanf(qtr,"%lf",&m_b3);
    if (v4==TRUE) fscanf(qtr,"%lf",&m_b4);
    fclose(qtr);
    m_sce=H_SCE1();
    m_mce=H_MCE();
    m_Femp=H_Femp();
    m_r2=H_R2();
    DeleteFile("temporal");
    CopyFile("Betas.dat","temporal",1);
    CopyFile("Confi.Agd","tempo",1);
}
//f
Proceso();
qtr=fopen("Confi.Agd","rt");
fscanf(qtr,"%lf",&tope);
fscanf(qtr,"%d",&v0);
fscanf(qtr,"%d",&v1);
fscanf(qtr,"%d",&v2);
fscanf(qtr,"%d",&v3);
fscanf(qtr,"%d",&v4);
fclose(qtr);
qtr=fopen("Betas.dat","rt");
if (qtr!=NULL) {
    if (v0==TRUE) fscanf(qtr,"%lf",&m_b0m);
    if (v1==TRUE) fscanf(qtr,"%lf",&m_b1m);
    if (v2==TRUE) fscanf(qtr,"%lf",&m_b2m);
    if (v3==TRUE) fscanf(qtr,"%lf",&m_b3m);
    if (v4==TRUE) fscanf(qtr,"%lf",&m_b4m);
    fclose(qtr);
    m_scem=G2_SCE1();
    m_mcem=G2_MCE();
    m_r2m=G2_R2();
    m_Fempm=G2_Femp();
}
//f
UpdateData(FALSE);
}
void CResultados2::OnGraerror() {
    Graerror2 Gr;
    Gr.DoModal();
}
void CResultados2::OnTamano() {
    CGrafico2 Gr;
    Gr.DoModal();
}
void CResultados2::OnOK() {CDialog::OnCancel();}

```


Archivo de encabezado Resultados2.h

```
#include "RoundButton.h"
#if
!defined(AFX_RESULTADOS2_H__AADE0214_2818_11D5_AB39_D1177D6EC063__INCLUDED_)
#define AFX_RESULTADOS2_H__AADE0214_2818_11D5_AB39_D1177D6EC063__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
class CResultados2 : public CDialog {
// Construction
public:
    CResultados2(CWnd* pParent = NULL); // standard constructor
    void Proceso(void);

// Dialog Data
   //{{AFX_DATA(CResultados2)
    enum { IDD = IDD_RESULTADOS2_DIALOG };
    CRoundButton    m_ok;
    CRoundButton    m_tama;
    CRoundButton    m_graerro;
    double    m_b0m;
    double    m_b0;
    double    m_b1m;
    double    m_b1;
    double    m_b2m;
    double    m_b2;
    double    m_b3m;
    double    m_b3;
    double    m_b4m;
    double    m_b4;
    double    m_Fempm;
    double    m_Femp;
    double    m_mcem;
    double    m_mce;
    double    m_r2m;
    double    m_r2;
    double    m_scem;
    double    m_sce;
    //}}AFX_DATA
// Overrides
   //{{AFX_VIRTUAL(CResultados2)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
protected:
   //{{AFX_MSG(CResultados2)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnGraerror();
    afx_msg void OnTamano();
    virtual void OnOK();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
#endif //
!defined(AFX_RESULTADOS2_H__AADE0214_2818_11D5_AB39_D1177D6EC063__INCLUDED_)
```

Archivo Grafico2.cpp

Gráfico de la tendencia del modelo seleccionado

```
#include "stdafx.h"
#include "AgMp.h"
#include "math.h"
#include "Grafico2.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CGrafico2::CGrafico2(CWnd* pParent /*=NULL*/) : CDialog(CGrafico2::IDD, pParent){
//{{AFX_DATA_INIT(CGrafico2)
// NOTE: the ClassWizard will add member initialization here
//}}AFX_DATA_INIT
}

void CGrafico2::DoDataExchange(CDataExchange* pDX){
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CGrafico2)
DDX_Control(pDX, IDOK, m_ok);
DDX_Control(pDX, IDC_Copiar, m_copiar);
DDX_Control(pDX, IDC_MSCHART1, m_Chart);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CGrafico2, CDialog)
//{{AFX_MSG_MAP(CGrafico2)
ON_WM_SHOWWINDOW()
ON_BN_CLICKED(IDC_Copiar, OnCopiar)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

long CGrafico2::numdatos(char ss[]) {
long contador=0;
double x;
FILE *qtr;
qtr=fopen("AgData.Dat", "rt");
if (qtr!=NULL) {
rewind(qtr);
while (!feof(qtr)) {
fscanf(qtr, "%lf", &x);
fscanf(qtr, "%lf", &x);
fscanf(qtr, "%lf", &x);
fscanf(qtr, "%lf", &x);
fscanf(qtr, "%lf", &x);
contador++;
}
}
fclose(qtr);
return contador;
}

void CGrafico2::OnShowWindow(BOOL bShow, UINT nStatus) {
```

```

COleSafeArray saRet;
FILE *qtr;
long iter=0;
long index[2];
double val,w,tope;
double Betas[5];
double temp[5];
int v[4],v2[4];
double x[4];
int p;
char aux[20];
CDialog::OnShowWindow(bShow, nStatus);
qtr=fopen("temporal","rt");
for (int i=0;i<5;i++) fscanf(qtr,"%lf",&temp[i]);
fclose(qtr);
qtr=fopen("tempo","rt");
fscanf(qtr,"%lf",&tope);
for (i=0;i<4;i++) fscanf(qtr,"%d",&v2[i]);
fclose(qtr);
qtr=fopen("Betas.Dat","rt");
for (i=0;i<5;i++) fscanf(qtr,"%lf",&Betas[i]);
fclose(qtr);
qtr=fopen("Confi.Agc","rt");
fscanf(qtr,"%lf",&tope);
for (i=0;i<4;i++) fscanf(qtr,"%d",&v[i]);
fclose(qtr);
qtr=fopen("n.dat","rt");
fscanf(qtr,"%d",&p);
fscanf(qtr,"%d",&p);
fclose(qtr);
iter=numdatos("AgData.Dat");
DWORD numElements[] = {iter,2};
saRet.Create(VT_R8, 2, numElements);
qtr=fopen("AgData.Dat","rt");
for(index[0]=0;index[0]<iter;index[0]++) {
    index[1]=0;
    fscanf(qtr,"%lf",&val);
    saRet.PutElement(index,&val);
    for (int i=0;i<4;i++) fscanf(qtr,"%lf",&x[i]);
    if (tope==0) {
        w=Betas[0];
        for (i=1;i<p;i++) {
            if (v[0]==TRUE) {
                w*=exp(Betas[i]*x[0]);
                v[0]=FALSE;
            }
            else if (v[1]==TRUE) {
                w*=exp(Betas[i]*x[1]);
                v[1]=FALSE;
            }
            else if (v[2]==TRUE) {
                w*=exp(Betas[i]*x[2]);
                v[2]=FALSE;
            }
            else if (v[3]==TRUE) {
                w*=exp(Betas[i]*x[3]);
                v[3]=FALSE;
            }
        }
    }
}
}

```



```

} //if
else {
    w=1/tope;
    for (i=1;i<p;i++) {
        if (v[0]==TRUE) {
            w*=pow(Betas[i],x[0]);
            v[0]=FALSE;
        } //if
        else if (v[1]==TRUE) {
            w*=pow(Betas[i],x[1]);
            v[1]=FALSE;
        } //if
        else if (v[2]==TRUE) {
            w*=pow(Betas[i],x[2]);
            v[2]=FALSE;
        } //if
        else if (v[3]==TRUE) {
            w*=pow(Betas[i],x[3]);
            v[3]=FALSE;
        } //if
    } //for
} //else
index[1]=1;
saRet.PutElement(index,&w);
if (tope==0) {
    w=temp[0];
    for (i=1;i<p;i++) {
        if (v2[0]==TRUE) {
            w*=exp(temp[i]*x[0]);
            v2[0]=FALSE;
        } //if
        else if (v2[1]==TRUE) {
            w*=exp(temp[i]*x[1]);
            v2[1]=FALSE;
        } //if
        else if (v2[2]==TRUE) {
            w*=exp(temp[i]*x[2]);
            v2[2]=FALSE;
        } //if
        else if (v2[3]==TRUE) {
            w*=exp(temp[i]*x[3]);
            v2[3]=FALSE;
        } //if
    } //for
} //if
else {
    w=1/tope;
    for (i=1;i<p;i++) {
        if (v2[0]==TRUE) {
            w*=pow(temp[i],x[0]);
            v2[0]=FALSE;
        } //if
        else if (v2[1]==TRUE) {
            w*=pow(temp[i],x[1]);
            v2[1]=FALSE;
        } //if
        else if (v2[2]==TRUE) {
            w*=pow(temp[i],x[2]);
            v2[2]=FALSE;
        } //if
    }
}

```

```

        }//if
        else if (v2[3]==TRUE) {
            w*=pow(temp[i],x[3]);
            v2[3]=FALSE;
        }//if
    }//for
    }//else
    saRet.PutElement(index,&w);
} //for
fclose(qtr);
m_Chart.SetChartData(saRet.Detach());
for (i=0;i<iter;i++) {
    m_Chart.SetRow(i+1);
    itoa(i,aux,10);
    m_Chart.SetRowLabel(aux);
} //for
m_Chart.Refresh();
}

void CGrafico2::OnCopiar() {m_Chart.EditCopy();}

```

Archivo de encabezado Grafico2.h

```
#include "RoundButton.h"
//{{AFX_INCLUDES()
#include "mschart.h"
//}}AFX_INCLUDES
#ifndef AFX_GRAFICO2_H_D12FAF42_13B1_11D5_AB39_8BA52D8B6D5D__INCLUDED_
#define AFX_GRAFICO2_H_D12FAF42_13B1_11D5_AB39_8BA52D8B6D5D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CGrafico2 : public CDialog{
// Construction
public:
    CGrafico2(CWnd* pParent = NULL); // standard constructor
    long numdatos(char ss[]);

// Dialog Data
   //{{AFX_DATA(CGrafico2)
    enum { IDD = IDD_GRAFICO2_DIALOG };
    CRoundButton    m_ok;
    CRoundButton    m_copiar;
    CMSChart        m_Chart;
    }//}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CGrafico2)
    protected:
        virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }//}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(CGrafico2)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnCopiar();
    }//}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
#ifndef AFX_GRAFICO2_H_D12FAF42_13B1_11D5_AB39_8BA52D8B6D5D__INCLUDED_
```

Archivo Graerror2.cpp

Gráfico de los residuos del modelo seleccionado

```
#include "stdafx.h"
#include "AgMp.h"
#include "Graerror2.h"
#include "math.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

Graerror2::Graerror2(CWnd* pParent /*=NULL*/)
: CDialog(Graerror2::IDD, pParent) {
//{{AFX_DATA_INIT(Graerror2)
// NOTE: the ClassWizard will add member initialization here
//}}AFX_DATA_INIT
}

void Graerror2::DoDataExchange(CDataExchange* pDX){
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(Graerror2)
    DDX_Control(pDX, IDOK, m_ok);
    DDX_Control(pDX, IDC_Copiar, m_copiar);
    DDX_Control(pDX, IDC_MSCHART1, m_Chart);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(Graerror2, CDialog)
    //{{AFX_MSG_MAP(Graerror2)
    ON_WM_SHOWWINDOW()
    ON_BN_CLICKED(IDC_Copiar, OnCopiar)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

long Graerror2::numdatos(char ss[]) {
    long contador=0;
    double x;
    FILE *qtr;
    qtr=fopen("AgData.Dat","rt");
    if (qtr!=NULL) {
        rewind(qtr);
        while (!feof(qtr)) {
            fscanf(qtr,"%lf",&x);
            fscanf(qtr,"%lf",&x);
            fscanf(qtr,"%lf",&x);
            fscanf(qtr,"%lf",&x);
            fscanf(qtr,"%lf",&x);
            contador++;
        }
    }
    fclose(qtr);
    return contador;
}

void Graerror2::OnShowWindow(BOOL bShow, UINT nStatus) {
```

```

COleSafeArray saRet;
FILE *qtr;
long iter=0;
long index[2];
double val,w,tope;
double Betas[5];
double temp[5];
int v[4],v2[4];
double x[4];
int p;
char aux[20];
CDialog::OnShowWindow(bShow, nStatus);
qtr=fopen("temporal","rt");
for (int i=0;i<5;i++) fscanf(qtr,"%lf",&temp[i]);
fclose(qtr);
qtr=fopen("tempo","rt");
fscanf(qtr,"%lf",&tope);
for (i=0;i<4;i++) fscanf(qtr,"%d",&v2[i]);
fclose(qtr);
qtr=fopen("Betas.Dat","rt");
for (i=0;i<5;i++) fscanf(qtr,"%lf",&Betas[i]);
fclose(qtr);
qtr=fopen("Confi.Agc","rt");
fscanf(qtr,"%lf",&tope);
for (i=0;i<4;i++) fscanf(qtr,"%d",&v[i]);
fclose(qtr);
qtr=fopen("n.dat","rt");
fscanf(qtr,"%d",&p);
fscanf(qtr,"%d",&p);
fclose(qtr);
iter=numdatos("AgData.Dat");
DWORD numElements[] = {iter,2};
saRet.Create(VT_R8, 2, numElements);
qtr=fopen("AgData.Dat","rt");
for(index[0]=0;index[0]<iter;index[0]++) {
    index[1]=1;
    fscanf(qtr,"%lf",&val);
    for (int i=0;i<4;i++) fscanf(qtr,"%lf",&x[i]);
    w=Betas[0];
    for (i=1;i<p;i++) {
        if (v[0]==TRUE) {
            w*=exp(Betas[i]*x[0]);
            v[0]=FALSE;
        }
        else if (v[1]==TRUE) {
            w*=exp(Betas[i]*x[1]);
            v[1]=FALSE;
        }
        else if (v[2]==TRUE) {
            w*=exp(Betas[i]*x[2]);
            v[2]=FALSE;
        }
        else if (v[3]==TRUE) {
            w*=exp(Betas[i]*x[3]);
            v[3]=FALSE;
        }
    }
    w=val-w;
    saRet.PutElement(index,&w);
}

```

```

        index[1]=0;
        w=temp[0];
        for (i=1;i<p;i++) {
            if (v2[0]==TRUE) {
                w*=exp(temp[i]*x[0]);
                v2[0]=FALSE;
            }
            else if (v2[1]==TRUE) {
                w*=exp(temp[i]*x[1]);
                v2[1]=FALSE;
            }
            else if (v2[2]==TRUE) {
                w*=exp(temp[i]*x[2]);
                v2[2]=FALSE;
            }
            else if (v2[3]==TRUE) {
                w*=exp(temp[i]*x[3]);
                v2[3]=FALSE;
            }
        }
        w=val-w;
        saRet.PutElement(index,&w);
    }
    fclose(qtr);
    m_Chart.SetChartData(saRet.Detach());
    for (i=0;i<iter;i++) {
        m_Chart.SetRow(i+1);
        itoa(i,aux,10);
        m_Chart.SetRowLabel(aux);
    }
    m_Chart.Refresh();
}

void Graerror2::OnCopiar() {m_Chart.EditCopy();}

```


Archivo de encabezado Graerror2.h

```
#include "RoundButton.h"
//{{AFX_INCLUDES()}
#include "mschart.h"
//}}AFX_INCLUDES
#if !defined(AFX_GRAERROR_H__AEC2C6C1_186D_11D5_AB39_813D7B8EFE62__INCLUDED_)
#define AFX_GRAERROR_H__AEC2C6C1_186D_11D5_AB39_813D7B8EFE62__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class Graerror2 : public CDialog{
// Construction
public:
    Graerror2(CWnd* pParent = NULL); // standard constructor
    long Graerror2::numdatos(char []);

// Dialog Data
    {{{AFX_DATA(Graerror2)
    enum { IDD = IDD_GRAERROR2_DIALOG };
    CRoundButton    m_ok;
    CRoundButton    m_copiar;
    CMSChart        m_Chart;
    }}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    {{{AFX_VIRTUAL(Graerror2)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }}}AFX_VIRTUAL

// Implementation
protected:
    {{{AFX_MSG(Graerror2)
    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
    afx_msg void OnCopiar();
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif //
!defined(AFX_GRAERROR_H__AEC2C6C1_186D_11D5_AB39_813D7B8EFE62__INCLUDED_)
```

BIBLIOGRAFIA

- Goldberg, D.E. (1989) "Genetic Algorithms in Search, Optimization and Machine Learning", Reading, Mass.: Addison- Wesley.
- Holland, J. (1975) "Adaptation in Natural and Artificial Systems", The University of Michigan Press, Ann Arbor.
- Dawkins, R. (1982) "The Extended Phenotype: The Gene as a Unit of Selection", Oxford Univ. Press, Oxford.
- Obitko, M. (1998) "Introduction to Genetic Algorithms", Alemania, University of Applied Sciences.
<http://cs.felk.cvut.cz/~xobitko/ga/>
- Herrán, M. (1995) "Vida Artificial", España, El grupo Gaia.
<http://www.geocities.com/SiliconValley/Vista/7491/>
- Contreras, J. (1996) "Operadores genéticos", Venezuela,
<http://www.iamnet.com/users/jcontre/genetic/operadores.htm>
- Larrañaga, P. (2000) "Introducción a los algoritmos genéticos",
<http://www.geocities.com/CapeCanaveral/9802/3d5ca000.htm>
- Ferrer, M. (1997) "Población y recursos naturales", Universidad de navarra, Departamento de humanidades biomedicas
<http://www.eumed.net/cursecon/2/index.htm>
- "Población, Conceptos Básicos"
<http://www.lafacu.com/>
- Preciado, J. (1999) "Futuro",
<http://www.pagina12.com.ar/1999/futuro.htm>
- "La ecología y los insectos", Perú, Universidad de Lamolina
<http://tarwi.lamolina.edu.pe/~acg/ç>
- Cordon J. (1999) "La ecuación logística, modelo poblacional", Dpto. de Matemáticas, Estadística y Computación (MATESCO)
<http://193.144.183.17/~cordon/doctor98/proyecto/logistic.htm>
- "Redes neuronales artificiales"
<http://www.gc.ssr.upm.es/inves/neural/ann2/introduc/intro.htm>