CAPITULO 4: DESARROLLO

En los capítulos anteriores hemos estudiado las bases del problema de la ruta más corta. Conocemos algunos métodos para resolverlo de los cuales hemos elegido algoritmos evolutivos, en particular SSPGA. Originalmente SSPGA se desarrollo para resolver problemas de la ruta más corta con distribuciones discretas en los costos de los arcos, sin embargo para cumplir el objetivo que se planteo originalmente, debemos reemplazar dichos costos discretos por continuos.

Para lograr este objetivo fue necesario, en primera instancia, recodificar SSPGA de C++ a Visual Basic. Por otro lado fue necesario también programar las funciones de distribución continuas con las cuales va a trabajar la nueva versión de SSPGA para después integrarlas al algoritmo original.

En el siguiente capitulo trata con la programación de la nueva versión de SSPGA denominada SSPEA. Empezando con la recodificación del algoritmo original, siguiendo con los generadores de funciones continuas para finalmente integrar dichas funciones al algoritmo.

4.1 Recodificación del Algoritmo Original.

Originalmente, SSPGA se codificó en C el primer paso en el desarrollo de este nuevo algoritmo, fue programar SSPGA en Visual Basic. La primera versión de SSPGA en Visual Basic se desarrolló en Visual Basic for Applications para Excel, sin embargo, bajo este ambiente se aumentaba el tiempo de ejecución, y se limitaba su funcionalidad a computadoras que contaran con este programa.

Para solucionar este problema, se decidió cambiar el ambiente a Visual Basic Studio. Si bien el lenguaje de programación es el mismo, este ambiente es más poderoso que el utilizado anteriormente y no es necesario contar con ningún software instalado para ejecutar los programas desarrollados con este lenguaje.

En esta segunda versión se realizo también un cambio en la manera de introducir los datos, mientras que en Excel las variaciones discretas en los costos de los arcos eran introducidas directamente desde el código (El usuario no podía modificarlas), mientras que en la segunda versión estos costos se introducen desde una ventana de ingreso de datos y el usuario puede seleccionar que distribución desea para cada nodo.

4.2 Introducción de Funciones Continuas.

Para generar cualquier distribución de probabilidad (continua o discreta), es necesario partir de un número aleatorio de una distribución uniforme continua y después transformarlo a la distribución que se desea simular. En este caso se utilizó el generador de números aleatorios de Visual Basic. A continuación se presentan las bases de los métodos utilizados para generar las distribuciones de probabilidad:

4.2.1 Distribución exponencial.

En el caso de la Exponencial con parámetro β, se utilizó el método de transformación inversa en el cual se genera un número aleatorio de U(0,1) (Uniforme sobre el intervalo 0,1) y utilizando la función de distribución se obtiene el valor de X [6].

4.2.2 Distribución gamma.

Los parámetros de esta distribución son α y β y para este caso, por simplicidad, se supondrá un α entero. Una de las propiedades de esta función es que, si α es entero, esta se puede escribir como la suma de α distribuciones Exponenciales independientes con parámetro β. Como anteriormente se definió un método para generar distribuciones exponenciales es conveniente utilizar esta propiedad y así simular esta distribución [6].

4.2.3 Distribución beta.

El procedimiento utilizado para generar un número aleatorio de la distribución Beta con parámetros α y β se le debe a Jöhnk, en este caso ninguno de los parámetros se restringe [6].

4.2.4 Distribución normal.

Sin duda la más conocida y utilizada de las distribuciones de probabilidad es la Normal. Existen una infinidad de métodos para generar un número aleatorio

proveniente de una Normal. Entre ellos se encuentra el método de Box y Müller el cual se basa en la relación entre una Normal (con media 0 y varianza 1) y una uniforme (0,1), dicho método se aplicó para simular esta distribución [6].

4.3 Pruebas de Hipótesis.

Para estar seguros de que los números aleatorios que se generen de cada distribución efectivamente provienen de la misma es necesario probar su confiabilidad. Para ello existen diferentes pruebas que se le pueden aplicar a muestras obtenidas de los generadores que se desarrollaron.

Todas los métodos descritos en el capitulo 4.2 provienen de transformaciones probadas matemáticamente, sin embargo estos se basan en la generación de un número aleatorio de una uniforme (0,1) producido por el generador de números aleatorios de Visual Basic. Por lo tanto la única prueba necesaria es para dicho generador.

Existen diferentes pruebas para números aleatorios, en este caso se utilizó la prueba Ji-Cuadrada para el generador de Uniforme(0,1). Aunque con esta prueba sería suficiente, también se probó la distribución Normal y Exponencial (Por ser las más representativas) con el programa estadístico MINITAB.

4.3.1 Prueba ji-cuadrada para uniforme (0,1)

La manera más fácil de probar si nuestro generador es confiable es producir una muestra de números y después ver que tanto se parecen a una distribución

Uniforme(0,1). Para se utilizó la prueba Ji-Cuadrada con parámetros conocidos (Pues conocemos los parámetros de la distribución).

La manera de hacer esta prueba es dividir el intervalo (0,1) en *k* sub-intervalos del mismo tamaño, luego se generamos $U_1,\ldots,U_n$ con nuestro generador. (Una regla empírica nos dice que *n/k* debe de ser al menos 5 y *k* no menor de 100). Además, para cualquier $j = 1,2,\ldots,k$, definimos f_j como el número de U_i que están en el j-esimo intervalo. Finalmente hagamos

$$X^2 = \frac{k}{n}\sum_{i=1}^{k}(f_j - \frac{n}{k})^2$$

De esta manera, para un *n* lo suficientemente grande, X^2 se distribuye Ji-Cuadrada con *k*-1 grados de libertad bajo la hipótesis nula de que los U_i efectivamente se distribuyen Uniforme (0,1). Entonces, rechazamos la hipótesis con un nivel α si $X^2 > X^2_{k-1,1-\alpha}$. Donde $X^2_{k-1,1-\alpha}$ es el punto critico superior de una distribución Ji-Cuadrada con *k-1* grados de libertad [7].

Se aplicó esta prueba a diez mil números aleatorios obtenidos de nuestro generador, utilizando 100 sub-intervalos de tamaño .1 y se obtuvieron los siguientes resultados:

Valor $X^2 = 94.32$

Valor $X^2_{99,1-.05} = 123.22$

Como podemos observar, $X^2 < X^2_{k-1,1-\alpha}$ ya que 94.32 < 123.22, por lo tanto no rechazamos la hipótesis y concluimos que efectivamente los datos obtenidos por el generador provienen de una Uniforme (0,1).

4.3.2 Pruebas para normal y exponencial en MINITAB.

Para el caso de una distribución Normal, MINITAB cuenta con tres diferentes tipos de pruebas: Anderson-Darling, Ryan-Joiner y Kolmogorov-Smirnov. Como ya se probó la efectividad del generador original de números aleatorios solo se aplicó la primera de estas pruebas utilizando una muestra de diez mil números. Las hipótesis nula de esta prueba es que los valores provienen de una Normal (0,1), por lo tanto si se obtiene un p-value < α rechazamos la hipótesis, de lo contrario concluimos que los números se generaron correctamente. Estos son los resultados arrojados por el paquete:

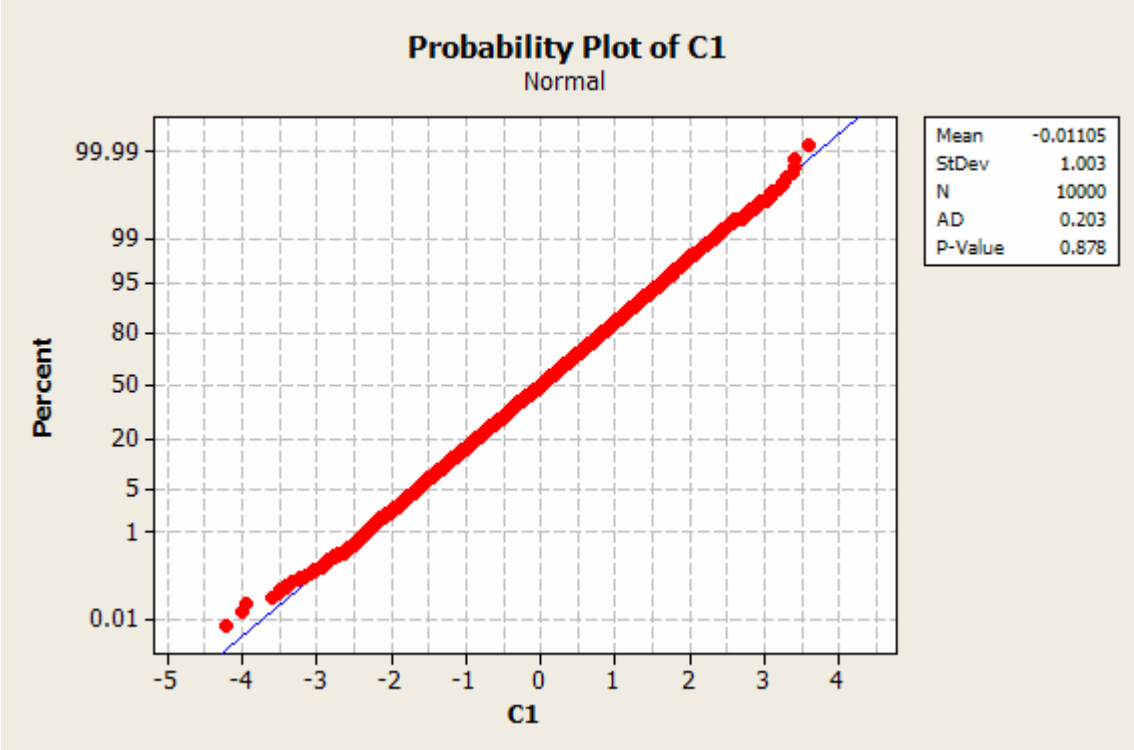


Figura 4.1 Resultados en MINITAB para prueba de Normal (0,1)

Si pedimos un α = .05 podemos observar claramente que p-value = .878 > α = .05, por lo tanto no rechazamos la hipótesis y podemos afirmar que lo datos provienen de una Normal con media 0 y varianza 1.

Así mismo, también se le hizo una prueba al generador de Exponencial ya que esta distribución se obtuvo por una transformación directa de la Uniforme(0,1). MINITAB tiene la opción de probar si una muestra proviene de alguna distribución que el usuario especifica. Se le aplico esta prueba diez mil números de nuestro generador Exponencial(1) y estos fueron los resultados obtenidos.

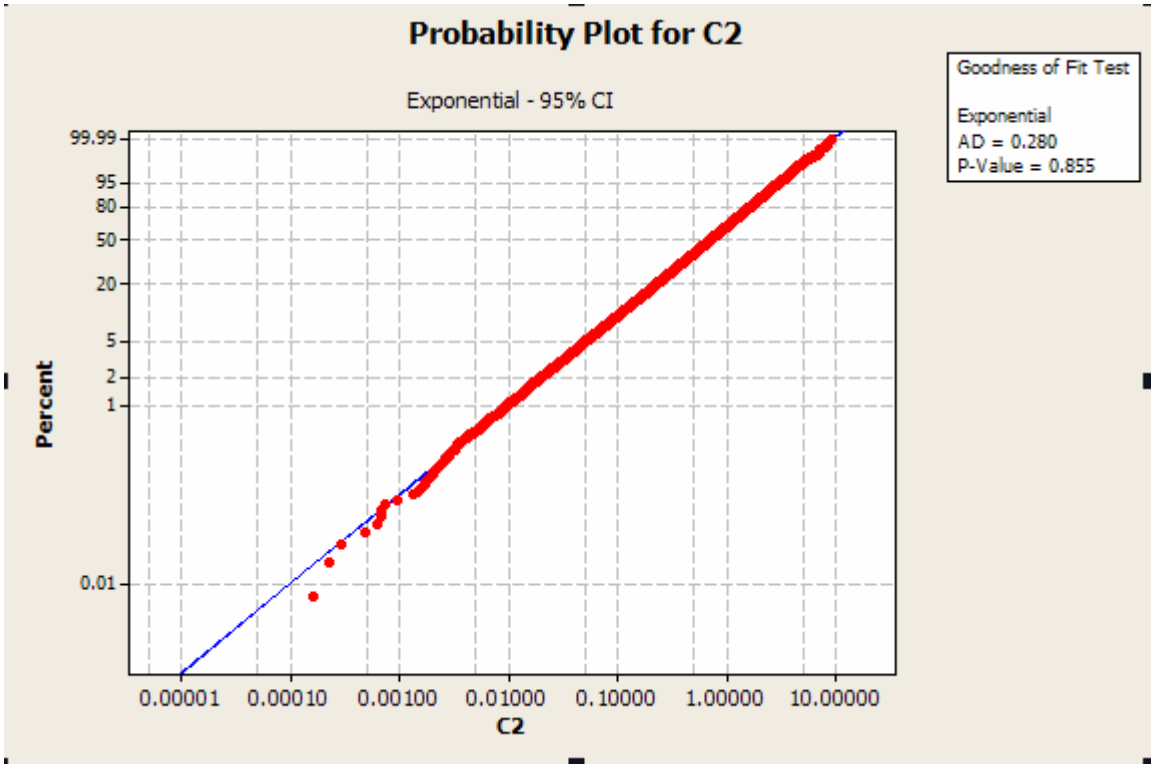


Figura 4.2 en MINITAB para prueba de Exponencial(1)

Para este caso obtuvimos un p-value = .855, utilizando $\alpha = .05$ igual que en la prueba anterior. Como la hipótesis nula es que los datos provienen de una distribución Exponencial(1) no rechazamos la hipótesis y concluimos que efectivamente provienen de la distribución Exponencial con parámetro 1.

4.3 Módulos del Programa.

Una vez que se tienen procedimientos codificados para las funciones continuas de distribución es necesario incorporarlos al programa recodificado para después combinarlo con la interfase visual para la captura de datos y despliegue de resultados. A continuación se presentan las partes principales del programa.

4.4.1 Entradas

Para ejecutar el programa es necesario que se introduzcan algunos datos necesarios para que algoritmo funcione. Es necesario conocer las características de la red que se quiere resolver, así como los parámetros de SSPGA (porcentajes de recombinación y mutación, número de generaciones y de corridas).

Al ejecutar el programa se le presenta al usuario una ventana principal como la que se presenta en la figura 4.3, en la cual pueden definir todos los datos que el algoritmo para ejecutarse, en la misma también se presentan los resultados una vez que termina la ejecución.

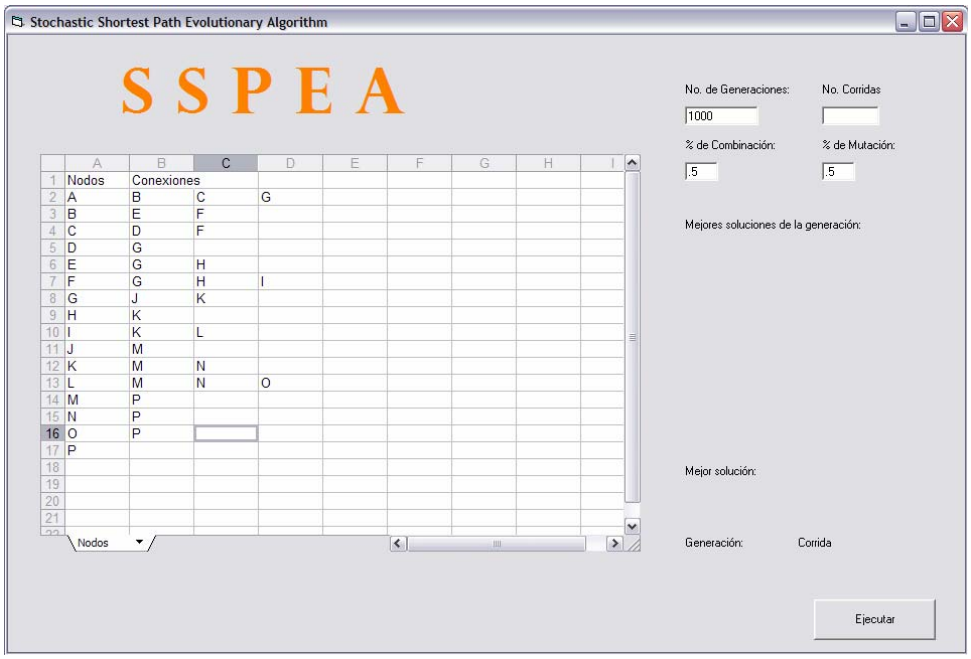
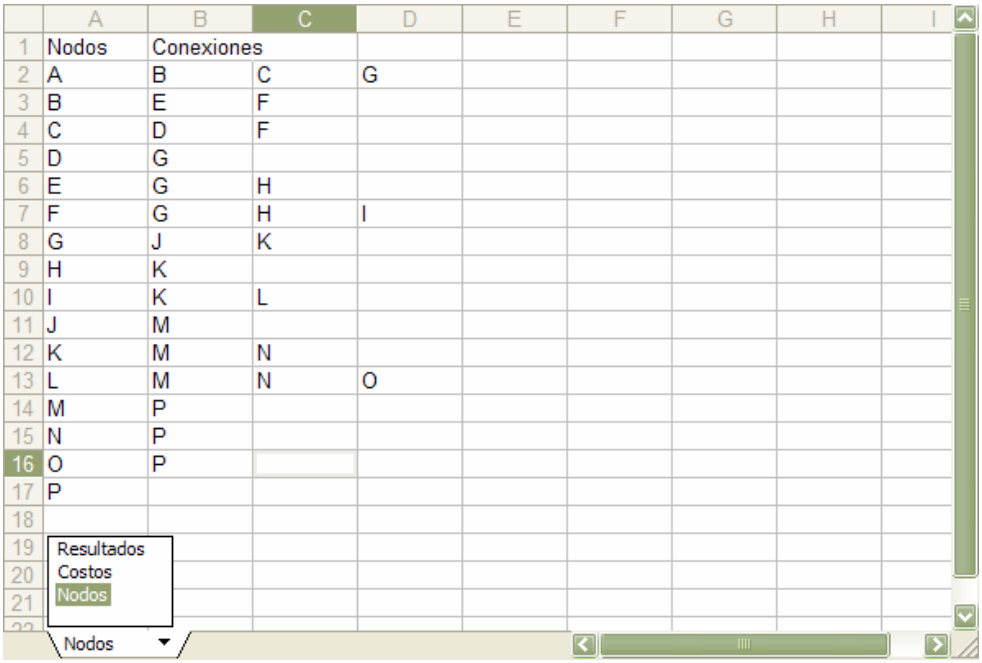


Figura 4.3 Ventana Principal.

En la ventana principal se puede encontrar una hoja de cálculo, en la cual se van a introducir los nodos de la red, sus conexiones y los costos de los arcos, también aquí se van a presentar parte de los resultados obtenidos al final de la ejecución como se explicará más adelante.



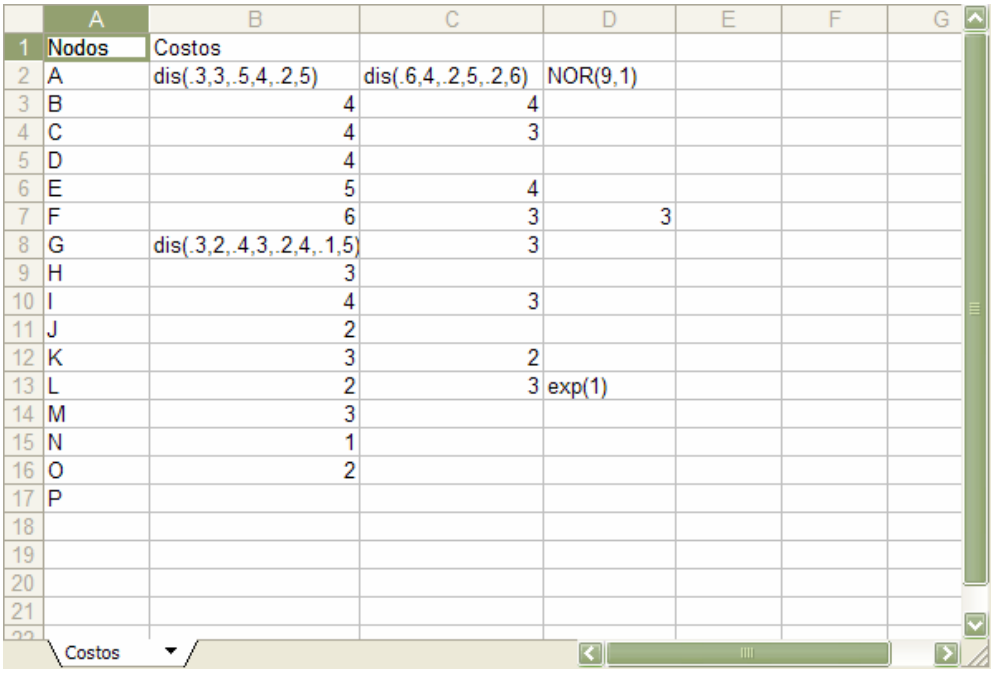
	A	B	C	D
1	Nodos	Conexiones		
2	A	B	C	G
3	B	E	F	
4	C	D	F	
5	D	G		
6	E	G	H	
7	F	G	H	I
8	G	J	K	
9	H	K		
10	I	K	L	
11	J	M		
12	K	M	N	
13	L	M	N	O
14	M	P		
15	N	P		
16	O	P		
17	P			

Figura 4.4 Hoja de Cálculo (Nodos)

La hoja de cálculo se divide en tres partes, una denominada “Nodos” ,otra denominada “Costo” y finalmente la hoja “Resultados. En “Nodos” se introducen los nodos y sus conexiones, mientras que en “Costo” entran los costos de cada arco (Constantes, Continuos o Discretos), la hoja de “Resultados” se utiliza al finalizar el algoritmo para el despliegue de los resultados.

En la Figura 4.4 se puede apreciar la hoja “Nodos”, aquí se introducen, en la primera columna, todos los nodos que contiene la ruta (a partir del Segundo renglón),

en las columnas subsecuentes se introducen las conexiones de cada nodo en el renglón correspondiente.



	A	B	C	D	E	F	G
1	Nodos	Costos					
2	A	dis(.3,3,.5,4,.2,5)	dis(.6,4,.2,5,.2,6)	NOR(9,1)			
3	B	4	4				
4	C	4	3				
5	D	4					
6	E	5	4				
7	F	6	3	3			
8	G	dis(.3,2,.4,3,.2,4,.1,5)	3				
9	H	3					
10	I	4	3				
11	J	2					
12	K	3	2				
13	L	2	3	exp(1)			
14	M	3					
15	N	1					
16	O	2					
17	P						
18							
19							
20							
21							

Costos

Figura 4.5 Hoja de Cálculo (Costos)

La hoja “Costos” se presenta en la figura 4.5. Aquí se pueden introducir los costos de cada arco en la misma posición en la que se especificaron las conexiones en la hoja “Nodos”.

Si se quiere introducir un valor constante para el costo del arco simplemente se escribe en la celda el valor numérico de dicho costo. Por otra parte, si se quiere introducir una distribución de probabilidad ya sea continua o discreta se hace de acuerdo a la siguiente nomenclatura:

Tabla 4.1 Nomenclatura para distribuciones de probabilidad

Distribución	Nomenclatura
Uniforme Discreta	Dis(p1,v1,p2,v2,…,pn,vn)
Exponencial	Exp(β)
Gamma	Gam(α, β)
Beta	Bet(α, β)
Normal	Nor(media, varianza)

La ventana principal cuenta también con cuatro casillas en las cuales se puede introducir el número de generaciones deseadas, los porcentajes de mutación y recombinación, así como el número de corridas que se desean ejecutar.

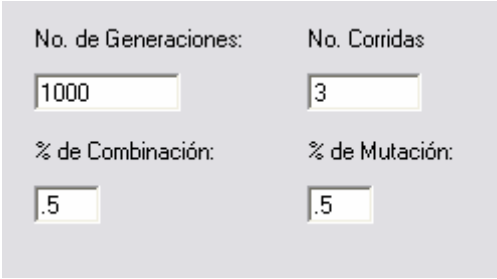


Figura 4.6 Entrada para parámetros.

Para la ejecución del algoritmo solo se debe hacer clic en el botón “Ejecutar”, ubicado en la parte inferior derecha de la Ventana Principal.

4.4.2 Cuerpo del programa.

El código principal del programa consta de 4 partes principales: la clase cromosoma, la clase Nodo, el módulo de funciones y el módulo Principal. Todas estas partes son necesarias para ejecutar paso a paso el algoritmo que previamente se definió.

Como se menciono anteriormente fue necesario definir 2 clases diferentes (cromosoma y nodo). Una clase es un conjunto de instrucciones y variables que componen un objeto, se pueden declarar cualquier cantidad de objetos de una clase.

Cuando se definió el problema y el funcionamiento de algoritmos genéticos, se mencionó la importancia tanto de los cromosomas como de los nodos, por esta misma importancia es por lo que las clases Nodo y Crosomoma son la columna vertebral de este programa.

La primera clase que se construyo fue Nodo, ya que a partir de ella se pueden construir los cromosomas, la población, etc. En esta parte del codigo se almacena toda la información correspondiente a los nodos, su nombre, sus conexiones y el costo de sus arcos, esto se hace por medio de los métodos “AgregaNodos” y “AgregaDist” respectivamente.

La propiedad más importante de un objeto definido como nodo es la capacidad de, por medo de la propiedad “ObtenDist”, encontrar el costo de cualquier arco interpretando la distribución del mismo y generando el número aleatorio correspondiente.

Una vez que se tienen los nodos es posible crear cromosomas con ellos, de ahí surge la clase cromosoma la cual se compone de objetos de tipo Nodo.

Esta clase cuenta con varias propiedades y métodos importantes. Primeramente cuenta con las propiedades “GetRuta” y “ObtenRuta”, con las cuales se le asigna al cromosoma una solución del Problema (un conjunto de Nodos que forman una ruta).

Esta solución se puede modificar con los métodos “Recombina”, el cual elimina el porcentaje de la ruta definido para la recombinación. Posteriormente esta ruta se puede completar con el método “LlenaAleatorio” tal y como se explicó en el marco teórico. Finalmente la clase cromosoma contiene un método que obtiene el costo total de la ruta haciendo referencia a la clase nodo, específicamente a la propiedad “ObtenDist” de cada nodo de la ruta.

En el módulo de funciones se programaron los métodos vistos anteriormente correspondientes a las funciones de distribución. Estos referenciados en la clase Nodo, cuando se busca el costo de algún arco.

Finalmente, en el modulo principal se integra todo para completar la estructura del programa. Aquí se define un vector que contiene objetos de tipo Cromosoma, a este compone la población del Algoritmo Genético. También en este modulo se encuentran los métodos para realizar la mutación con el porcentaje correspondiente y el ordenamiento de la población.

Todo el programa se ejecuta desde una rutina principal denominada “main”. Como ya se tienen los métodos para todas las partes del algoritmo, en esta rutina solo se llaman en el orden correspondiente, repitiendo el proceso el número de Generaciones que el usuario determine.

Debido a la aleatoriedad de las distribuciones de probabilidad continuas, es necesario ejecutar el algoritmo varias veces para poder observar con más precisión el comportamiento de las mejores rutas que se obtienen al final de la corrida ya que es posible que la mejor solución obtenida sea poco factible. En la versión final de SSPEA es posible determinar cuantas corridas se quieren ejecutar como se observa en la figura 4.6.

Finalmente en ocasiones es necesario ejecutar el algoritmo más de una vez para ver lo que sucede con la población. Por esta razón el programa incluye la opción correr el algoritmo varias veces de manera independiente guardando los resultados correspondientes de cada población.

4.4.3 Salidas

Cuando el programa termina de ejecutarse y el algoritmo esta completo, es hora de presentar los resultados al usuario. El despliegue de estos se realiza en la misma pantalla principal.

Parte de los resultados se van presentando dinámicamente mientras el programa se esta ejecutando, en la ventana principal se pueden observar las 5 mejores rutas de cada generación, así como la mejor solución encontrada hasta el momento. Esto se despliega para cada una de las corridas que se eligieron, en la parte inferior de la figura 4.6 se observa el número de la corrida que se esta ejecutando y la generación actual.

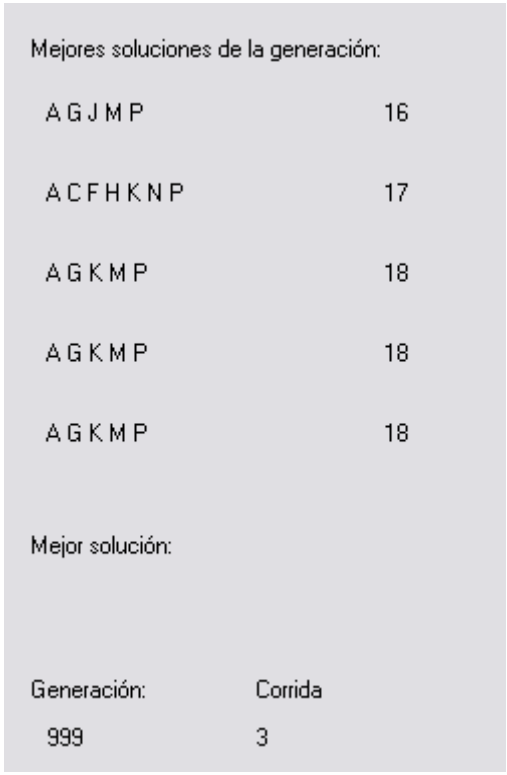
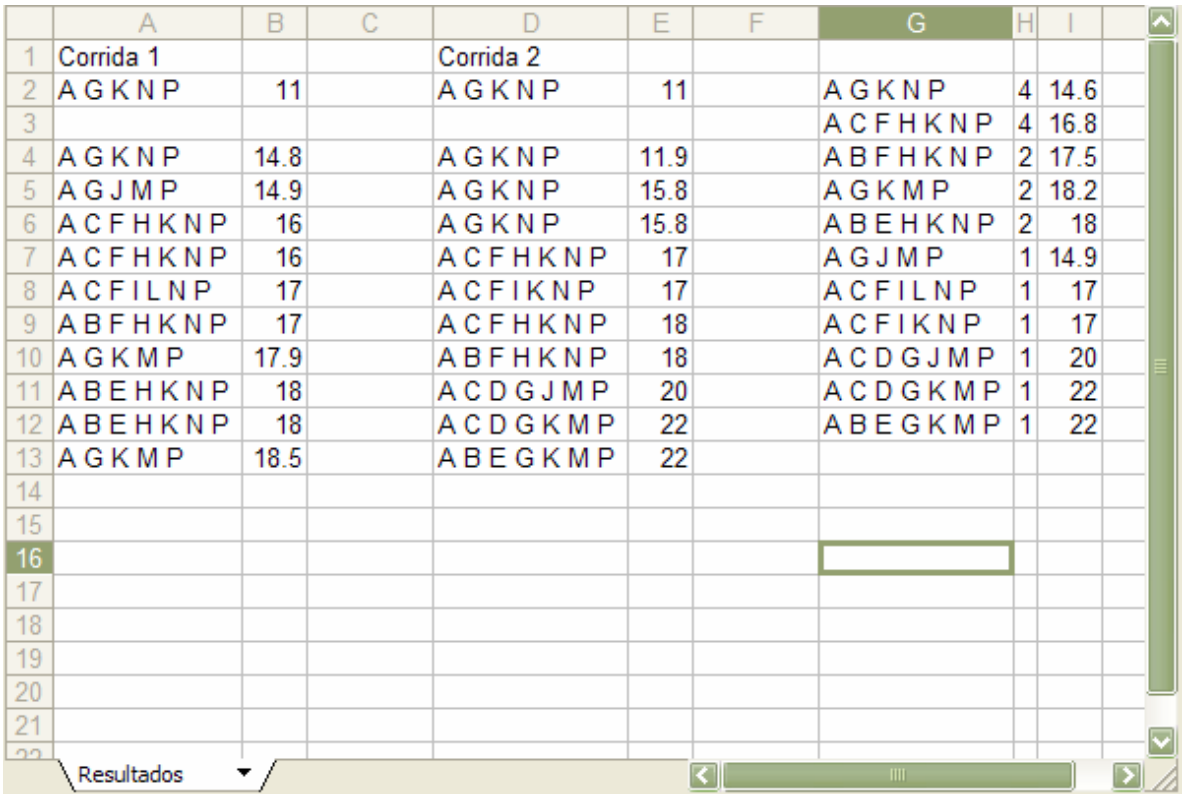


Figura 4.7 Despliegue de Resultados por generación

Una vez que el programa termina su ejecución se despliega un resumen en una hoja de cálculo adicional. En dicha hoja se pueden observar los resultados finales de la población en cada una de las corridas. Esto se muestra claramente en la figura 4.7.



	A	B	C	D	E	F	G	H	I
1	Corrida 1			Corrida 2					
2	A G K N P	11		A G K N P	11		A G K N P	4	14.6
3							A C F H K N P	4	16.8
4	A G K N P	14.8		A G K N P	11.9		A B F H K N P	2	17.5
5	A G J M P	14.9		A G K N P	15.8		A G K M P	2	18.2
6	A C F H K N P	16		A G K N P	15.8		A B E H K N P	2	18
7	A C F H K N P	16		A C F H K N P	17		A G J M P	1	14.9
8	A C F I L N P	17		A C F I K N P	17		A C F I L N P	1	17
9	A B F H K N P	17		A C F H K N P	18		A C F I K N P	1	17
10	A G K M P	17.9		A B F H K N P	18		A C D G J M P	1	20
11	A B E H K N P	18		A C D G J M P	20		A C D G K M P	1	22
12	A B E H K N P	18		A C D G K M P	22		A B E G K M P	1	22
13	A G K M P	18.5		A B E G K M P	22				
14									
15									
16									
17									
18									
19									
20									
21									

Resultados

Figura 4.8 Hoja “Resultados”

Para cada corrida se despliega la mejor ruta y su costo, así como el estado final de la población. Así mismo se presenta un resumen de las rutas que aparecieron al final de las dos corridas, cuantas veces aparecieron y el promedio de su rendimiento.

Los resultados se presentan de esta manera para tener más información acerca de lo que sucedió durante la ejecución del programa. Debido a la aleatoriedad de las funciones continuas, no es suficiente con conocer la mejor solución que arrojó la corrida, pues esta puede ser improbable, se deben de analizar la frecuencia con la que aparecieron las rutas y su calidad. En el capítulo siguiente se desarrollara con más detalle el análisis e interpretación de los resultados.