UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES

Centro De Ciencias básicas

Ingeniería en Computación Inteligente

3°A

"DOCUMENTACIÓN: Algoritmo Genético Simple"

Profesor: Dr. Alejandro Padilla Diaz

E Q U I P O

Elías Del Hoyo César Eduardo 262045

Rivera Delgadillo Ximena 261261

Sandoval Pérez José Luis 261731

SINTESIS

Los algoritmos genéticos desde 1975 se han caracterizado por representar soluciones a un determinado problema abordado en forma de cadena de bits.

Este merito se le otorga a John Halland que propuso este tipo de algoritmos, la popularización de estos radica en su eficacia y sencillez de implementación. Dentro de esta documentación explicaremos a detalle el funcionamiento de un algoritmo genético básico a través de una población de 10 individuos con 10 bits individualmente. En este algoritmo usaremos la mutación y cruzamiento de bits para poder llegar a el objetivo de nuestro algoritmo genético; tener a cada individuo de nuestra población con 10 bits en 0’s.

El algoritmo se desarrolló en el lenguaje C++ en 3 etapas denominadas de la siguiente manera:

1. Cruzamiento entre individuos.
2. Cruzamiento entre individuos y ordenamiento.
3. Cruzamiento entre individuos, ordenamiento y mutación.

FASE 1

a) ¿Qué casos (procesos) del Algoritmo Genético usaste para garantizar la evolución de todos los individuos con el proyecto?

Como se mencionó con anterioridad el algoritmo genético fue realizado en 4 etapas y con 4 procesos distintos. Estos procesos son definidos de la siguiente manera

1. Cruzamiento entre individuos:

   Este proceso del algoritmo genético es el más sencillo de los 3 y es la base para el correcto funcionamiento de nuestro algoritmo. En él, comenzamos con una población de 10 individuos, cada individuo con 10 bits *aleatorios*, los bits se reducen a 0's u 1's.

   Una vez teniendo la población a cada individuo se le asigna otro individuo de la misma población con el que cruzara sus bits, la selección de la cruza es la siguiente 1-10, 2-9, 3-8, 4-7, 5-6. Después en cada cadena de bits generamos un numero aleatorio del cual se partirá la cadena para hacer el cruzamiento, el cruzamiento resultante serán 2 hijos por cada cruzamiento, una vez teniendo los hijos y padres definidos contabilizamos los 0's de cada cadena y si los hijos tienes más 0's que los padres estos se convierten en las posiciones de los padres dentro de la población.

Ejemplificando tenemos lo siguiente:

Cadena 1: 0111100101
Cadena 10: 1111010101

Ahora el número de manera arbitraria tomaremos el numero 4 para hacer el cruzamiento de estas 2 cadenas(padres):

Cadena 1: 0101100101 4 – 0's
Cadena 10: 1111010101 3 – 0´s
Hijo 1-10: 0101010101 5 – 0´s
Hijo 10-1: 111100101 3- 0's

Como podemos notar el hijo de 1-10 tiene más 0's que la cadena 1(padre 1) y que el hijo de 10-1 tiene la misma cantidad de 0's que la cadena 10(padre 2), sabiendo esto y, tomando en cuenta la explicación anterior, lo que tenemos que hacer es regresar el hijo 1-10 a la posición de la cadena 1 dentro de nuestra población. Esto se realizará hasta que ya hayamos hecho los 5 cruzamientos mencionados anteriormente.

2. Ordenamiento de la población con individuo con mayor numero de 0's: tomando como base la etapa anterior al hacer cada cruzamiento tenemos que ordenar nuestra población, el ordenamiento se basara en el individuo con el mayor número de 0´s en la primera posición. Este proceso se repite hasta que encontremos un individuo tenga todos sus bits en 0´s. Tomando en cuenta que

el máximo de generaciones es 250. Cada generación se define al hacer los 5 cruzamientos.

3. Cruzamiento de individuos de manera aleatoria y ordenamiento de población:

   De la etapa 2 tenemos que hacer la siguiente modificación, ahora en vez de realizar 5 cruzamientos de la manera que se indicó, tenemos que seleccionar 6 individuos de manera aleatoria de 1-10 (tamaño de la población) y los 2 primeros números al azar los cruzaremos de manera sucesiva los 2 siguientes y los 2 siguientes. En cada cruza tenemos que ordenar de mayor a menos numero de 0's en el individuo. De nuevo los cruces y ordenamiento terminara hasta que se haya encontrado un individuo con 10 bits 0's.

4. Mutación de últimos 5 individuos de manera aleatoria:

   Teniendo ya las 3 etapas ahora cerraremos nuestros procesos con la mutación. La mutación se define al hacer un cambio de bit dentro de nuestro individuo, es decir, si un bit del individuo es 0 se cambia a 1 y si el bit es 1 se cambia a 0. Como se mencionó anteriormente la mutación solo puede realizarse en individuos en la posición 5-10 de la población. Para mutar se seleccionarán 2 individuos dentro de estas posiciones (20% de nuestra población) para mutar. Se generará 1 número aleatorio que indicara cuantos bits de la cadena se mutaran de 1-4(10% - 40%), estos bits también aleatorios. Ejemplificando tenemos:

   Cadena 7: 01**1**1100**1**01 mutación de 2 bits.

Cadena 10:1111010101 mutación de 1 bit.

Cadena 7 mutada: 0101100001
Cadena 10 mutada: 1011010101

Si un individuo de los que se pueden mutar contiene 10 0's este individuo ya no se podrá mutar. Al hacer la mutación se repetirá el proceso anterior y la mutación hasta que toda nuestra población de 10 individuos tenga sus bits en 0's. Con un máximo de 250 generaciones. (Cada generación termina después de mutación).

b) **¿Por qué se detecta precisamente un problema si no ordenas o no generas una evolución de los individuos, y qué sucede con el problema por resolver o mejorar con el CRUZAMIENTO, ORDENAMIENTO DE LOS MEJORES Y LA MUTACION? ¿Cuál es la justificación?**

Principalmente al no cruzar los individuos de la población implica que la única forma para encontrar un individuo con 10's 0's en que se genere de manera aleatoria y esto es demasiado difícil incluso casi imposible, se necesitaría demasiadas generaciones. Ahora al no ordenar nuestros individuos aumentarían nuestras generaciones porque la evolución de nuestros individuos sería un poco más lenta, al ordenar aumentamos la posibilidad de que en el siguiente cruce los hijos de los padres a cruzar contengan mayor cantidad de 0's. La situación con la mutación es similar, si no mutamos el proceso de evolución de la población es más largo, sin embargo, al mutar aumentamos las posibilidades para que la población sea pura, es decir, todos sus individuos con 10 bits 0's.

FASE 2

(Desarrollo de informe de casos)

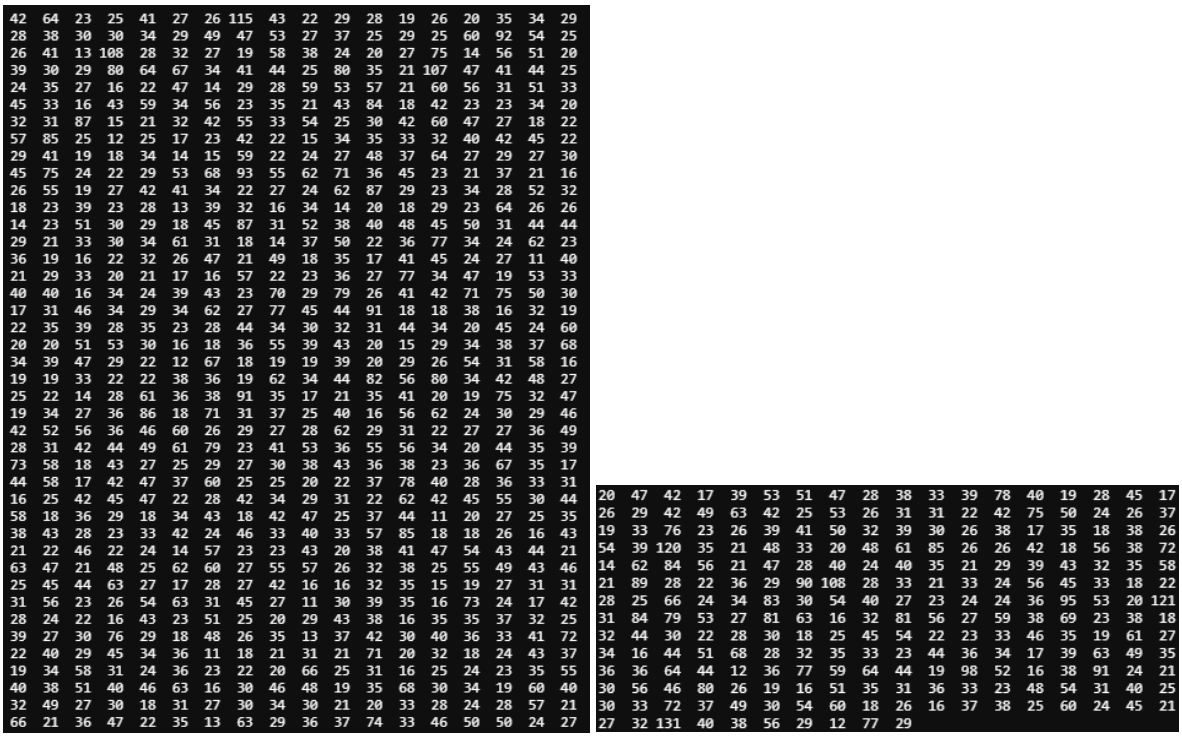
Formulación o propuesta

Para reportar la eficiencia de nuestro algoritmo genético se realizó un análisis exhaustivo del mismo, realizando 1000 corridas. La eficiencia de cada corrida se medirá en el número de generaciones poblacionales que genera el algoritmo hasta llegar a la generación pura 10 individuos con 10 bits 0's.

Análisis de resultados

Tenemos aquí los resultados obtenidos de dos programas, realizados por nosotros, donde se calculó un total de 1000 corridas (en cada uno):

PROGRAMA 1

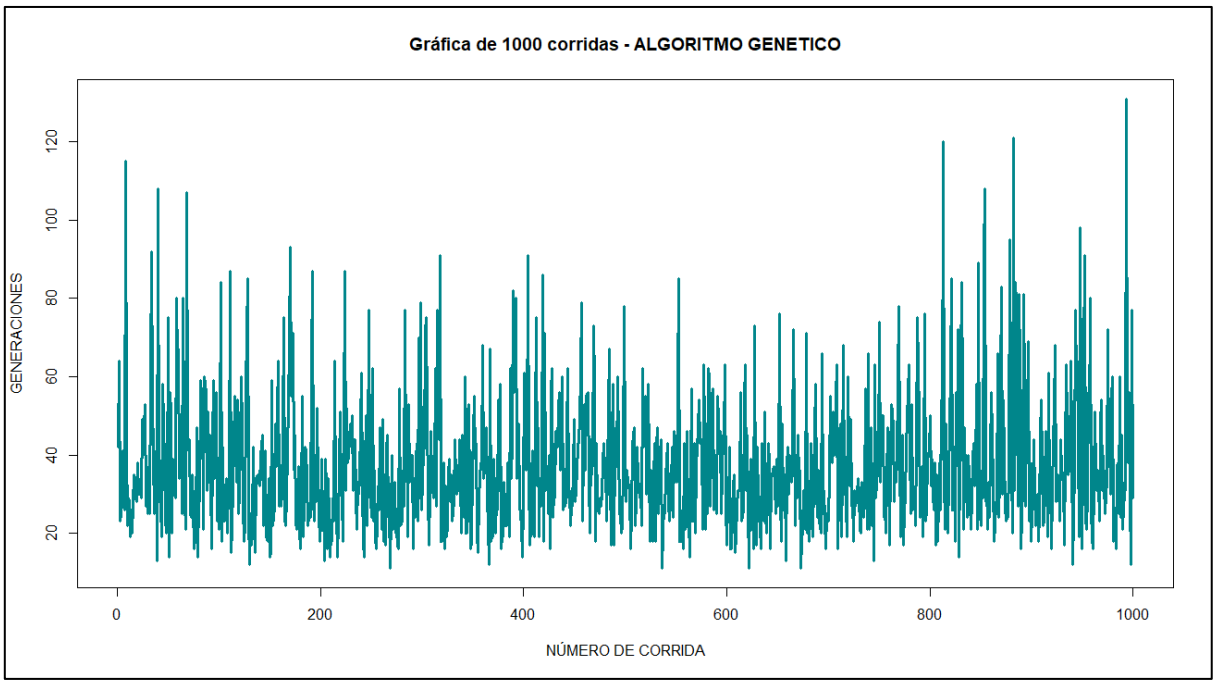


42	64	23	25	41	27	26	115	43	22	29	28	19	26	20	35	34	29
28	38	30	30	34	29	49	47	53	27	37	25	29	25	60	92	54	25
26	41	13	108	28	32	27	19	58	38	24	20	27	75	14	56	51	20
39	30	29	80	64	67	34	41	44	25	80	35	21	107	47	41	44	25
24	35	27	16	22	47	14	29	28	59	53	57	21	60	56	31	51	33
45	33	16	43	59	34	56	23	35	21	43	84	18	42	23	23	34	20
32	31	87	15	21	32	42	55	33	54	25	30	42	60	47	27	18	22
57	85	25	12	25	17	23	42	22	15	34	35	33	32	40	42	45	22
29	41	19	18	34	14	15	59	22	24	27	48	37	64	27	29	27	30
45	75	24	22	29	53	68	93	55	62	71	36	45	23	21	37	21	16
26	55	19	27	42	41	34	22	27	24	62	87	29	23	34	28	52	32
18	23	39	23	28	13	39	32	16	34	14	20	18	29	23	64	26	26
14	23	51	30	29	18	45	87	31	52	38	40	48	45	50	31	44	44
29	21	33	30	34	61	31	18	14	37	50	22	36	77	34	24	62	23
36	19	16	22	32	26	47	21	49	18	35	17	41	45	24	27	11	40
21	29	33	20	21	17	16	57	22	23	36	27	77	34	47	19	53	33
40	40	16	34	24	39	43	23	70	29	79	26	41	42	71	75	50	30
17	31	46	34	29	34	62	27	77	45	44	91	18	18	38	16	32	19
22	35	39	28	35	23	28	44	34	30	32	31	44	34	20	45	24	60
20	20	51	53	30	16	18	36	55	39	43	20	15	29	34	38	37	68
34	39	47	29	22	12	67	18	19	19	39	20	29	26	54	31	58	16
19	19	33	22	22	38	36	19	62	34	44	82	56	80	34	42	48	27
25	22	14	28	61	36	38	91	35	17	21	35	41	20	19	75	32	47
19	34	27	36	86	18	71	31	37	25	40	16	56	62	24	30	29	46
42	52	56	36	46	60	26	29	27	28	62	29	31	22	27	27	36	49
28	31	42	44	49	61	79	23	41	53	36	55	56	34	20	44	35	39
73	58	18	43	27	25	29	27	30	38	43	36	38	23	36	67	35	17
44	58	17	42	47	37	60	25	25	20	22	37	78	40	28	36	33	31
16	25	42	45	47	22	28	42	34	29	31	22	62	42	45	55	30	44
58	18	36	29	18	34	43	18	42	47	25	37	44	11	20	27	25	35
38	43	28	23	33	42	24	46	33	40	33	57	85	18	18	26	16	43
21	22	46	22	24	14	57	23	23	43	20	38	41	47	54	43	44	21
63	47	21	48	25	62	60	27	55	57	26	32	38	25	55	49	43	46
25	45	44	63	27	17	28	27	42	16	16	32	35	15	19	27	31	31
31	56	23	26	54	63	31	45	27	11	30	39	35	16	73	24	17	42
28	24	22	16	43	23	51	25	20	29	43	38	16	35	35	37	32	25
39	27	30	76	29	18	48	26	35	13	37	42	30	40	36	33	41	72
22	40	29	45	34	36	11	18	21	31	21	71	20	32	18	24	43	37
19	34	58	31	24	36	23	22	20	66	25	31	16	25	24	23	35	55
40	38	51	40	46	63	16	30	46	48	19	35	68	30	34	19	60	40
32	49	27	30	18	31	27	30	34	30	21	20	33	28	24	28	57	21
66	21	36	47	22	35	13	63	29	36	37	74	33	46	50	50	24	27

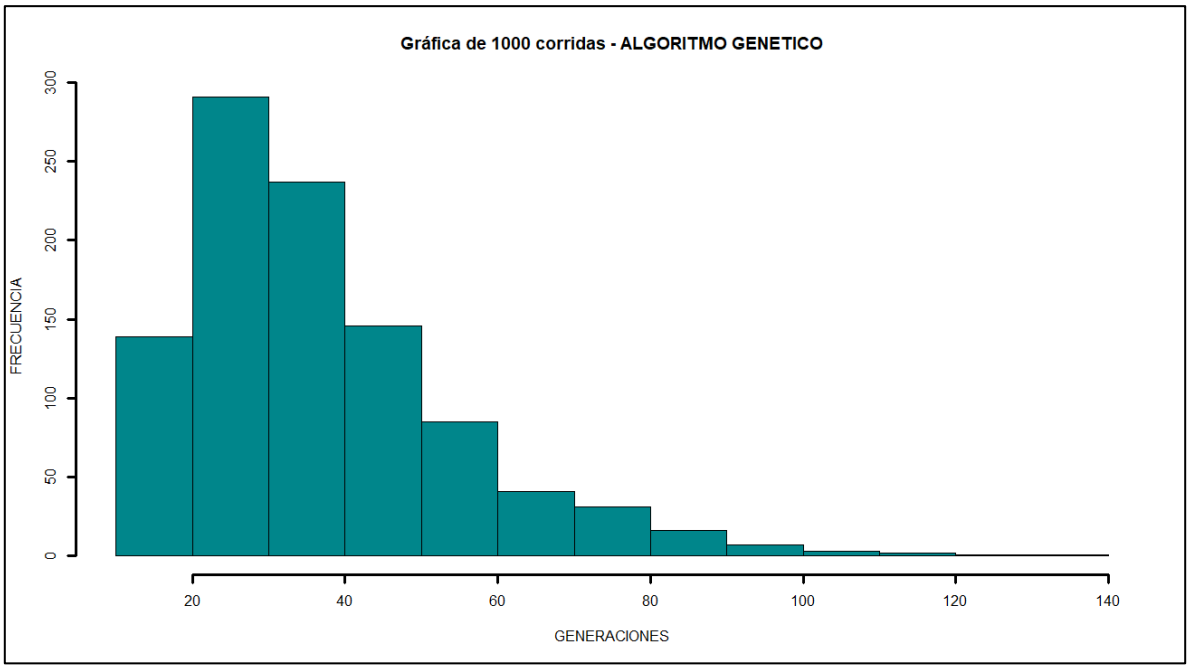
20	47	42	17	39	53	51	47	28	38	33	39	78	40	19	28	45	17
26	29	42	49	63	42	25	53	26	31	31	22	42	75	50	24	26	37
19	33	76	23	26	39	41	50	32	39	30	26	38	17	35	18	38	26
54	39	120	35	21	48	33	20	48	61	85	26	26	42	18	56	38	72
14	62	84	56	21	47	28	40	24	40	35	21	29	39	43	32	35	58
21	89	28	22	36	29	90	108	28	33	21	33	24	56	45	33	18	22
28	25	66	24	34	83	30	54	40	27	23	24	24	36	95	53	20	121
31	84	79	53	27	81	63	16	32	81	56	27	59	38	69	23	38	18
32	44	30	22	28	30	18	25	45	54	22	23	33	46	35	19	61	27
34	16	44	51	68	28	32	35	33	23	44	36	34	17	39	63	49	35
36	36	64	44	12	36	77	59	64	44	19	98	52	16	38	91	24	21
30	56	46	80	26	19	16	51	35	31	36	33	23	48	54	31	40	25
30	33	72	37	49	30	54	60	18	26	16	37	38	25	60	24	45	21
27	32	131	40	38	56	29	12	77	29								

Aquí observamos los resultados de las generaciones obtenidas en 1000 corridas. Podemos observar que los resultados son muy variados donde obtuvimos valores mínimos como 11 generaciones y un valor máximo de 131 generaciones.

El promedio obtenido fue de **37.203.** Esto nos ayuda a analizar que el programa genera la matriz de ceros en un aproximado de 37 generaciones. A continuación, mostramos una gráfica con los valores dados del algoritmo.

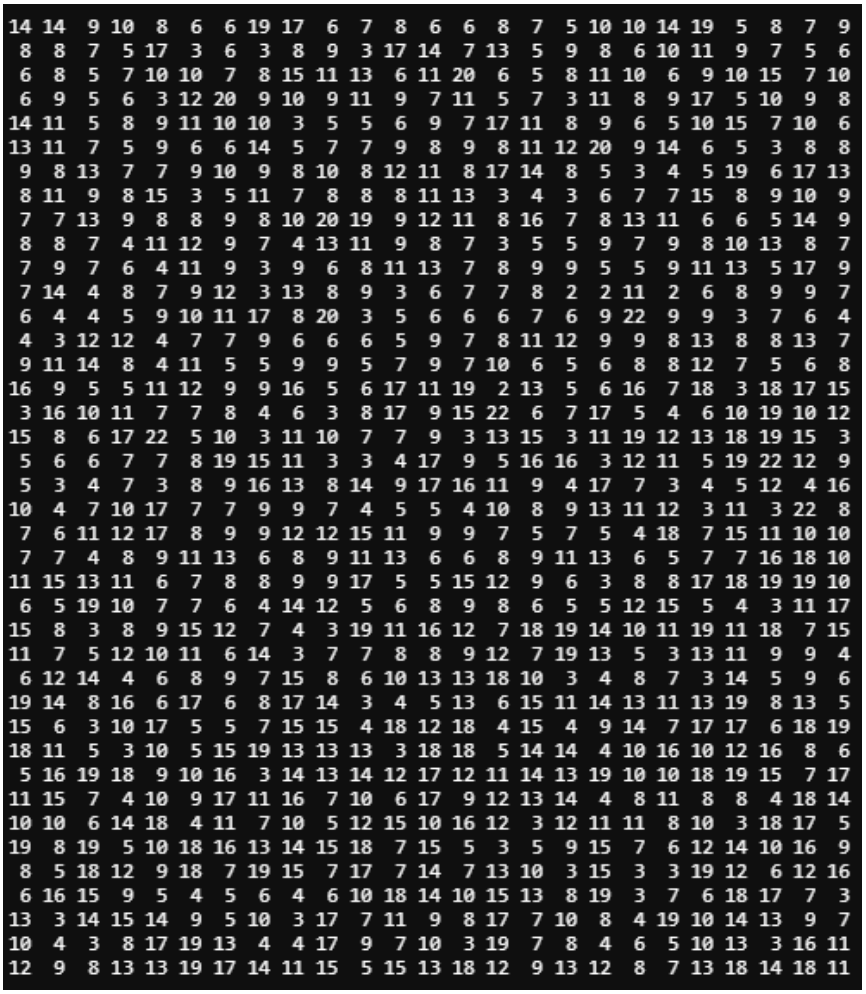


Podemos observar en esta gráfica los puntos en y (Generaciones) que se obtienen para cada corrida. Observamos que la mayoría de las corridas se encuentran dentro de las generaciones 20-60, aunque hay casos mayores como ciertos picos de la gráfica.



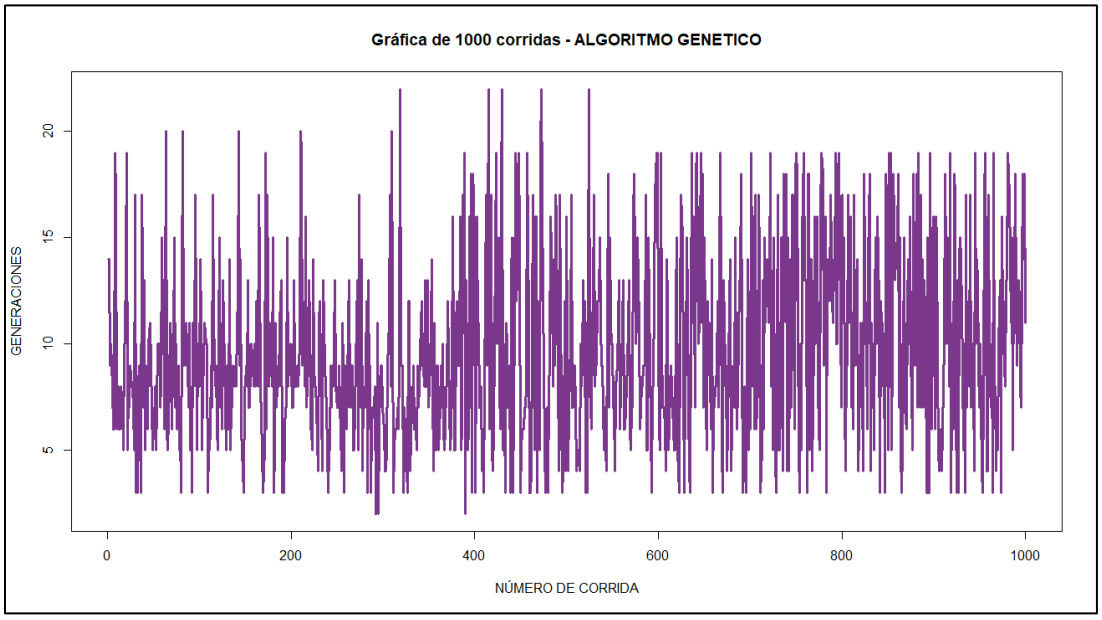
En este histograma podemos observar la frecuencia con la que se obtuvo cada generación, donde las generaciones de 20-30 son las que predominan con un total de casi 300 corridas. A pesar de que el promedio se encuentra en 37 corridas, esto no significa que sea el rango (30-40) con mayor frecuencia, pero al haber valores que sobrepasan las 80 generaciones, permite que el promedio sea mayor.

PROGRAMA 2

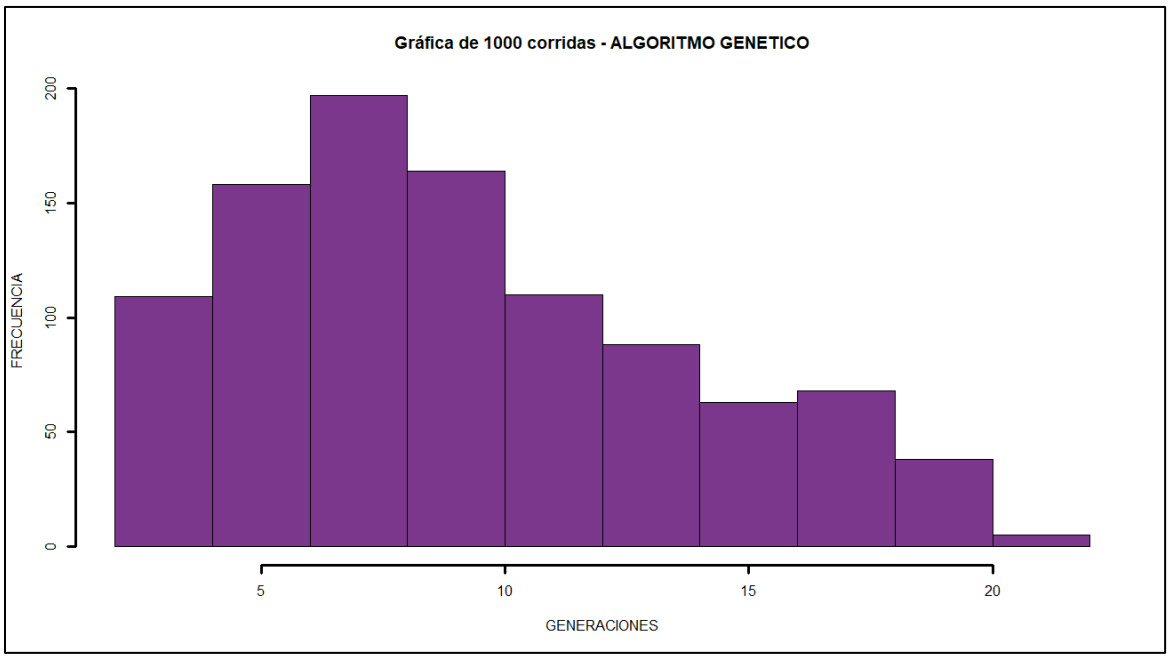


En este segundo programada realizado, observamos como la frecuencia en las generaciones obtenidas es de muy bajo rango, por lo que obtenemos un promedio de **9.68** generaciones por cada

corrida. Esto nos da a entender que este segundo programa permite obtener la generación máxima de una forma más rápida y eficaz en un menor tiempo de generaciones.



Esta primera gráfica nos muestra los valores dados en cada corrida. Para este programa las generaciones siempre se dan menores a 25 los cuales nos permiten tener un mejor control de los resultados, pues no se observan generaciones muy altas que alteren el promedio.



Por último, tenemos el histograma con las frecuencias de las generaciones obtenidas donde el predominante es entre las generaciones 6 y 8, obteniendo un total de casi 200 repeticiones. Esto nos demuestra que efectivamente es un algoritmo muy rápido que permite calcular la matriz máxima en un promedio menor a las 10 generaciones.

Descripción del proceso

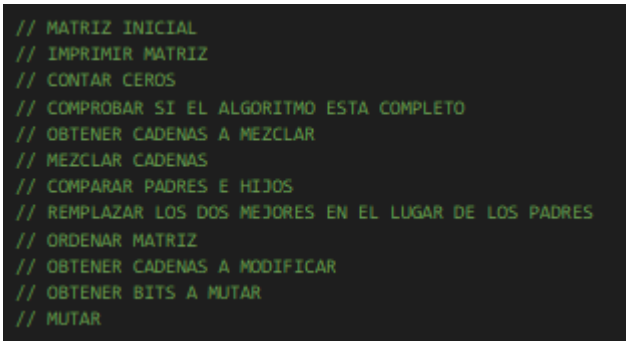
En la última etapa del proyecto, el cruzamiento era entre 6 individuos seleccionados de manera aleatoria; la mutación era únicamente para los últimos 5 individuos de la población siempre y cuando ninguno de estos tuviera los 10 ceros. De estos existía la posibilidad de mutar de 0-4 bits los cuales eran seleccionados de manera aleatoria. Dentro de la cadena, al aumentar los bits a mutar aumentaba la posibilidad de que, al cruzar la cadena mutada con una cadena pura, la cadena mutada se convirtiera en pura.

A su vez, teníamos un condicional que nos decía lo siguiente: si se obtenía un total de 9 cadenas llenas de 0, las mutaciones dejaban de realizarse y esa única cadena restante se cruzaría hasta completar la matriz de 0 máxima. Esto también permitió que, al llegar a 9 cadenas de ceros, fuera más fácil obtener esa última cadena y que el algoritmo encontrara su solución más rápido.

Las cadenas se ordenaban después de mutarlas, después de eso se repetía el patrón, cruce, retorno de hijos, mutación, ordenamiento de cadena con mayor numero de 0's. Así sucesivamente hasta completar el algoritmo.

Narración de los casos

Para este programa seguimos la siguiente secuencia (en este caso basada el programa 1 realizado):



- El programa comienza creando una matriz inicial con valores 0 y 1 de manera al azar.
- Después imprime la matriz y hace un conteo de si alguna cadena contiene los 10 ceros.
- Previo a la mezcla y mutación, se debe comprobar si el algoritmo ya está completo, de lo contrario procedemos a mezclar. Para ello se realiza el conteo de ceros previamente y así podemos saber si el algoritmo a finalizado.

- Proseguimos a obtener la separación aleatoria de la cadena, nos servirá para mezclar posteriormente
- Para mezclar, primeramente, se seleccionan las 6 cadenas padres aleatorias para cruzar, y claramente se hacen los cruces aleatorios.
- Ya obtenida la separación y las cadenas para mezclar, pasamos a mezclarlas.
- Ya tenidas las mezclas, mediante la ayuda de una matriz temporal, comparamos y ordenamos las dos cadenas resultantes (hijos) con las dos cadenas seleccionadas para la mezcla (padres) donde, se seleccionarán las dos mejores y se colocarán en el espacio de las dos cadenas padres
- Se ordena la matriz y se hace conteo de ceros obtenidos después de los cruces
- Ahora proseguimos con la mutación. Para ello se tiene un condicional que permita saber si se tienen 9 cadenas de ceros; de no ser así, se continua el proceso. Primeramente, se seleccionan las 2 cadenas a mutar: estas deben ser entre las últimas 5 cadenas y, en caso de haber cadenas con ceros, no se pueden seleccionar para la mutación
- Ya obtenidas esas cadenas, se obtiene un porcentaje seleccionado entre 10 y 40 % que permitirá saber si mutar de 1 a 4 bits (según sea el porcentaje). De acuerdo con el porcentaje, tomar aleatoriamente la cantidad de bits necesarios para mutar.
- Ya seleccionados los bits a mutar, únicamente se hace el intercambio: los bits 0, se cambian a 1 y viceversa
- Acabado este proceso, se ordena nuevamente la matriz para dar paso a la siguiente generación y realizar todo el proceso previamente descrito

Con este seguimiento, se logró realizar dicho algoritmo de manera que siempre está en constante mezcla y mutación, cumpliendo con los condicionales ya mencionados. Así mismo, podemos decir que los pasos a seguir fueron establecidos por el maestro, solamente seguimos las instrucciones y, además, agregamos algunos detalles de estética para que algoritmo muestre detalladamente cada paso realizado en el proceso.

FASE 3

(Resolución de los casos)

- **Análisis de las sentencias obtenidas**

De manera general través del programa desarrollado pudimos comprender mucho mejor el funcionamiento de los algoritmos genéticos, seguir pasos organizados para dar solución a un problema específico y trabajar con una técnica de programación inspirada en la reproducción de los seres vivos y que imita a la evolución biológica como estrategia para resolver problemas de optimización.

En conclusión, a partir de las sentencias obtenidas podemos comprender que un algoritmo genético consiste en pocas palabras de 4 etapas.

La primera que titulamos "Inicialización" donde se genera aleatoriamente una población inicial como los dice su nombre, constituida por posibles soluciones del problema, en este caso los individuos.

Una segunda que podemos llamar "Evaluación", donde le aplicamos la función de evaluación a cada uno de los individuos, es decir obtenemos las sentencias de los individuos y de la población inicial.

A partir de esto surgen las hipótesis de evolución ya que al ya conocer a profundidad la generación inicial y a sus individuos, podemos realizar un análisis de la población para saber que tan cerca o lejos se encuentra del punto final o de la optimización que buscamos.

Una siguiente etapa que reconocemos como evolución donde aplicamos los operadores genéticos como son selección, reproducción y mutación.

Para finalmente detener el Algoritmo genético cuando se alcance la solución óptima se alcance.

En relación con las sentencias obtenidas podemos observar que hay diferentes maneras de realizar una misma secuencia de pasos, y que, aunque estas partan de la misma problemática y obtengan el mismo resultado. Es de suma importancia la

comparación de tiempos de ejecución, espacios en memoria y claramente también en este caso en particular en cuantas Generaciones se llegaba al resultado esperado.

A lo largo del desarrollo del proyecto realizamos un sinfín de intentos de programas, versiones optimizadas y sin optimizar, algunas con mayor tiempo de ejecución que otras y con menos. Lo cual se convirtió en una parte clave de nuestra experiencia al avance de nuestro proyecto, ya que nos permitió generar diferentes perspectivas y formas de realizar lo mismo.

▪ ¿Qué estrategias usaron para forzar la obtención o mejora de los 10 individuos y por qué?

Algunas de las estrategias que implementamos para la obtención o mejora de los 10 individuos fueron el cruce entre estos y la mutación de bits.

Condicionales como que si un individuo ya se encontraba en el punto esperado ya no se le realizara el proceso de mutación y únicamente experimentara con el resto de la población de 10 individuos, para de esta forma ya no dañara la población mejorada y únicamente enfocar la mutación en aquellos individuos que aún se encontraban fuera de la sentencia de contener puros bits de 0.

Estas estrategias nos permitieron generar una red de límites para manipular la población e identificar a partir de cuales individuos se podían continuar las funciones del algoritmo genético y desde cuales ya no.

Del mismo modo aquí podemos también recalcar que una de las funciones más importantes del algoritmo era la correcta implementación de la función de ordenamiento ya que de acuerdo con esta podíamos crear limites muy importantes de entrada y salida a las otras funciones.

▪ ¿Qué pudo haber funcionado mejor y por qué? Recomendaciones y lecciones aprendidas

Experimentando con el código y las diferentes formas de realizar este. Logramos identificar algunos errores y lecciones, sobre que pudimos haber mejorado. Entre estas, cosas aparentemente sencillas como la implementación de funciones, estructuras de datos, ciclos y librerías para la facilitación de tareas. Igualmente, también puntos más elaborados de la producción del código como las condicionales de las mutaciones

de los individuos, donde podíamos limitar la mutación de individuos para forzar o ayudar a los individuos a llegar al resultado esperado

- **¿Cuáles recomendaciones pueden realizarse en este proyecto?**

Algunas de las recomendaciones que daríamos para la realización de este proyecto serian la organización de funciones y de ideas iniciales, generar un ordenamiento en el desarrollo de funciones y de condicionales a implementar desde un comienzo del programa, ir más allá de las indicaciones o de las propuestas del Algoritmo e implementar condicionales, ciclos y funciones para la optimización del programa y de las generaciones.

- **¿Cuáles son las lecciones aprendidas en el proyecto?**

Gracias a este proyecto pudimos implementar algunos de nuestros conocimientos recolectados a lo largo de la carrera en Algoritmos Genéticos que se enmarcan en los algoritmos evolutivos, que incluyen también las estrategias evolutivas de la programación evolutiva y la programación genética.

Pudimos comprender con mayor profundidad el porqué de lo poderoso y exitoso del Algoritmo Genético para resolver problemas, y de la misma forma el poder de los principios evolutivos.

Por qué se han utilizado algoritmos genéticos en una amplia variedad de campos para desarrollar soluciones a problemas tan difíciles.

De esta manera concluimos que los algoritmos genéticos pueden tener muchas aplicaciones en diferentes ámbitos de la vida cotidiana y como estas aplicaciones a través de la inteligencia artificial mejoran cada día ayudándonos a descubrir cosas nuevas y nuevas posibilidades de desarrollo de problemas.