UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS
VERITATE SOLA NOBIS IMPONETUR VIRILIS TOGA. 1948

Facultad de Ingeniería Eléctrica

Departamento de Electro energética

Trabajo de Diploma

Título: Algoritmos genéticos mejorados para la reconfiguración de sistemas eléctricos de distribución

Autor: Alexis García Cárdenas
Tutor: Dr. José Ángel González Quintero

Santa Clara
2015

"Año 57 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Algoritmos genéticos mejorados para la reconfiguración de sistemas eléctricos de distribución

Autor: Alexis García Cárdenas

Tutor: Dr. José Ángel González Quintero
E-mail: pepe@uclv.edu.cu

Santa Clara

2015

"Año 57 de la Revolución"

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Automática, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Autor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

Pensamiento

La verdadera ciencia no suprime nada, sino que busca las cosas que no comprende y las mira cara a cara, sin turbarse. Negar los hechos no los suprime, como cerrar los ojos.

Paul Bernard (1866-1947) novelista francés.

DEDICATORIA

Dedico este trabajo a todos los que me apoyaron a lo largo de estos años de estudios, y en especial a mi familia que siempre ha estado ahí para mí.

AGRADECIMIENTOS

Agradezco a toda mi familia por su apoyo, y en especial a mi mamá por todos sus esfuerzos y preocupaciones.
A los profesores y maestros que han estado ahí para regalarme un poco de su conocimiento.
A mi tutor por su ayuda en la realización de este trabajo

TAREA TÉCNICA

- Hacer un estudio del estado del arte sobre el tema de los AG aplicados al problema de reconfiguración.
- Proponer mejoras a los AG aplicados a la reconfiguración.
- Realizar pruebas con casos concretos.

Firma del Autor	Firma del Tutor

RESUMEN

En este trabajo se tratan algunos de los algoritmos evolutivos y métodos heurísticos más utilizados en la actualidad, también se ofrece una pequeña descripción de los mismos, y se orienta en dirección de los AGs a los cuales se les hace una descripción general y se les implementa en la interfaz del MATLAB, con el cual se realizan una serie de corridas a casos concretos de ejemplos de reconfiguración de sistemas de distribución para demostrar su valía a la hora de resolver este tipo de problemática mejorando algunos de sus parámetros para disminuir el tiempo de las corridas.

Índice

PENSAMIENTO i

DEDICATORIA ii

AGRADECIMIENTOS iii

RESUMEN v

INTRODUCCIÓN 1

Introducción: ..... **¡Error! Marcador no definido.**

CAPÍTULO 1. Estudio del arte 3

1.1. 1.1 Métodos Heurísticos 3

1.2. 1.2 Computación Evolutiva. 4

1.3. 1.3 Método del Simple Intercambio de Ramas. 5

1.4. 1.4 Patrón de Flujo Óptimo. 5

1.5. 1.7 Búsqueda Tabú. 6

1.6. 1.8 Redes Neuronales. 6

1.7. 1.9 Optimización de Colonias de Hormigas. 7

1.8. 1.10 Optimización de Enjambre de Partículas. 8

1.9. 1.11 Recocido Simulado (SA) 9

1.10. 1.12 Algoritmos Genéticos. 10

1.10.1 1.12.1 Orígenes de los Algoritmos Genéticos. 11

1.10.2 1.12.2 Como saber cuándo usar los Algoritmos Genéticos. 12

1.10.3 1.12.3 Algunas Ventajas y Desventajas de los Algoritmos Genéticos. 12

1.10.4 1.12.4 Algoritmo Genético Simple. 14

1.10.5 1.12.5 Algoritmos Genéticos Paralelos (AGP). 15

1.10.6 1.12.6Algoritmos maestro-esclavo: 15

1.10.7 1.12.7 Comparación de los Algoritmos Genéticos con otras técnicas heurísticas. 17

CAPÍTULO 2. MATERIALES Y MÉTODOS (u otro nombre de capítulo).....18

1.11. 2.1Algoritmo de creación de la población:.....19

1.12. 2.2Creación de la Población Inicial.....20

1.12.1 2.1.1Generar aleatoriamente cada circuito o individuo de la población inicial. 21

1.12.2 2.1.2Algoritmo de Prim 21

1.12.3 2.1.3Codigo de implementación del algoritmo por el método de Generar aleatoriamente cada individuo de la población inicial:.....25

1.13. 2.1.4Creación de la población inicial mediante operaciones de mutación.....25

1.13.1 2.1.5Codigo de implementación del algoritmo por el método de Creación de la población inicial mediante operaciones de mutación: 26

1.14. 2.1.6Generar una población genéticamente variada. 28

1.15. 2.2Algoritmo de Creación de la Población Inicial 29

1.16. 2.3Optimizacion de individuos de la población: 31

CAPÍTULO 3. RESULTADOS Y DISCUSIÓN (u otro nombre de capítulo).....37

1.1. 3.1Codigos empleados en la comparación de métodos:.....37

1.2. 3.2Ejemplos de casos concretos.....38

1.2.1 3.2.1Para circuito de 30 barras: 38

1.2.2 3.2.2Para circuitos de 57 barras: 52

1.2.3 3.2.3Para circuito de 118 barras: 69

CONCLUSIONES Y RECOMENDACIONES 86

Conclusiones .... **¡Error! Marcador no definido.**

Recomendaciones 86

REFERENCIAS BIBLIOGRÁFICAS 87

ANEXOS 89

Anexo I Diagrama de circuito IEEE de 30 barras. 89

INTRODUCCIÓN

Al ser las redes de distribución la parte más extensa de los sistemas eléctricos, se ha hecho necesario analizar algunas variables, tales como el efecto capacitivo, los límites de voltaje, la cantidad de carga instalada, y otras más. Ahora bien, en este trabajo, lo más importante es lograr menores índices de pérdidas a la hora de buscar la reconfiguración más eficiente. Actualmente todos los procesos tienden a ser automatizados para lograr una mejor eficacia de los procesos industriales, de ahí que gracias a este desarrollo no sea necesario ir probando con cada interruptor hasta alcanzar los resultados aproximadamente deseados, ya que la tecnología actual permite emplear *softwares* que ayudan a simular las diferentes situaciones que pueden acontecer, y a partir de estos implementar algoritmos y métodos computacionales que permitan alcanzar un resultado bastante cercano a una solución óptima. Estos algoritmos y métodos son muchos y variados, ejemplo de ello son: el método del Simple intercambio de rama, el Patrón de Flujo Optimo, la Búsqueda Tabú, la Optimización de enjambre de partículas, la Optimización de colonias de hormigas, la Programación Evolutiva de objetivo múltiple, los Algoritmos Genéticos, entre otros muchos. Aquí se mencionarán solo los que usan métodos heurísticos y algoritmos evolutivos, pero la clasificación de los mismos es mucho mayor, y la cantidad de campos de la tecnología actual que se dedican al estudio de métodos de cálculos

aproximados para la resolución de problemas crecen cada día. Puesto que la tecnología y la electricidad están tan profundamente relacionadas entre sí, el uso del MATLAB para ayudar a con la resolución de circuitos puede ser de mucha ayuda, es por eso que este trabajo tiene como objetivo general perfeccionar las características de los AG aplicados al problema de la reconfiguración de redes eléctricas de distribución mediante la mejora de la variabilidad genética y el tiempo de creación de la población inicial para aumentar la probabilidad de éxito en la búsqueda del circuito óptimo. Y para dar cumplimiento al mismo se trazaron los siguientes objetivos específicos:

- Revisión del estado del arte.

- Describir y plantear el Algoritmo.

- Implementación del Algoritmo en MATLAB. Corrida de casos concretos.

Organización del informe:

La estructura del trabajo se dividió en tres capítulos siguiendo los objetivos específicos enunciados:

En el Primer Capítulo: "Algoritmos evolutivos y métodos heurísticos más utilizados", se mencionan algunos de los algoritmos evolutivos y métodos heurísticos que más se usan en la actualidad, y se brinda una pequeña descripción de los mismos.

En el Segundo Capítulo: "Implementación de los Algoritmos Genéticos", se brinda una descripción de las mejoras propuestas a los AGs, y se explica el AG utilizado para dichas mejoras.

En el Tercer Capítulo: "Resultados y Análisis de casos", se muestran los resultados obtenidos después de correr el AG con las nuevas mejoras.

CAPÍTULO 1. Estudio del arte

La reconfiguración de las redes de distribución es una temática delicada. Las mismas son de gran extensión, y a pesar de ser diseñadas de forma mallada para realizar transferencias de cargas en el momento que sea necesario, estas son operadas de forma radial para facilitar la operación de los sistemas de protección y la detección de fallas. Por tanto no tener en cuenta la reconfiguración radial más óptima produciría un incremento sustancial de las pérdidas de potencia y la no fiabilidad del sistema. De ahí se le dedique tanto esfuerzo, ejemplo de ello son los muchos métodos y algoritmos que son creados mediante el uso de la programación evolutiva y métodos heurísticos. Algunos de ellos son: el método del Simple Intercambio de Rama, Patrón de Flujo Optimo, la Búsqueda Tabú, Optimización de Enjambre de Partículas, Optimización de Colonias de Hormigas, la Programación Evolutiva Multi-Objetivo, las Redes Neuronales, el Recocido Simulado, y los Algoritmos Genéticos. Aunque existen otros, solo se mencionarán los antes citados, no porque sean los más importantes sino solo por poner algunos ejemplos en concreto.
En este capítulo se explicarán algunos de los métodos mencionados encargados del surgimiento de todos los algoritmos de optimización.

1.1. 1.1 Métodos Heurísticos.

En su sentido más amplio, un algoritmo heurístico será un algoritmo para un problema de optimización basado en una estrategia (idea) transparente (habitualmente sencilla) para buscar dentro del conjunto de soluciones factibles y que no presenta ninguna garantía de encontrar el óptimo. En un sentido un poco

más específico, un heurístico es un algoritmo para el que no hay ningún resultado matemático que garantice que lleva a soluciones bastante buenas en un tiempo razonable, pero cuya idea parece asegurar un buen comportamiento para la mayoría de los ejemplos del problema en cuestión. [1, 2]

1.2. Computación Evolutiva.

La computación evolutiva utiliza métodos heurísticos, y copia los mecanismos de evolución natural, expuestos por la teoría de Charles Darwin, recogida en su obra *El Origen de las Especies por medio de la Selección Natural: la selección natural, la reproducción y la diversidad genética de individuos.* En la computación evolutiva se crea una población compuesta por variables que representan posibles soluciones a un determinado problema que se quiere resolver, y simulando el proceso de evolución estas van mejorando mediante el cruzamiento y la mutación, y van siendo comparadas con soluciones anteriores, y con funciones de aptitud, de forma que se desechan los menos óptimos, y entonces iterativamente se obtiene una solución que se acerque lo más posible a la solución deseada. Muchos han sido los modelos de computación evolutiva creados, y hay que decir que todos en mayor o menor medida como se ha dicho anteriormente, están compuestos en general por:

1) Una representación o codificación de las soluciones potenciales al problema bajo estudio.
2) Una población (conjunto de individuos) de estas soluciones potenciales.
3) Mecanismos para generar nuevos individuos o soluciones potenciales al problema estudiado, a partir de los miembros de la población actual (los denominados operadores de mutación y recombinación).
4) Una función de desempeño o evaluación (del inglés *fitness function*) que determina la calidad de los individuos en la población en su capacidad de resolver el problema bajo estudio.
5) Un método de selección que otorgue mayores probabilidades de sobrevivir a las buenas soluciones.

Luego de haber aclarado estos conceptos se pasará a presentar ejemplos concretos de ellos. [3]

1.3. 1.3 Método del Simple Intercambio de Ramas.

La idea básica del método de intercambio de rama heurístico es calcular el cambio de pérdidas de potencia operando un par de interruptores (cerrando uno y abriendo el otro al mismo tiempo), la meta final es reducir las pérdidas de potencia. En este proceso las líneas de la red pueden ser consideradas como interruptores por sí mismas [2].

Para utilizar este método se propone seguir el algoritmo siguiente:

1) Obtener una red inicial de configuración radial.

2) Obtener el conjunto de interruptores abiertos (y cerrados) que se manipularán.

3) Correr un flujo de cargas para todas las posibles combinaciones de este conjunto tomando nota de sus pérdidas de potencia y del cumplimiento de las restricciones impuestas.

4) Mostrar los resultados de la configuración topológica con menor *función objetivo* (Pérdidas de Potencia) y que cumple con las restricciones impuestas al problema.

La ventaja de este método radica en lo fácil que resulta su programación y su simplicidad para producir siempre variantes radiales. Pero presenta algunas desventajas, por ejemplo la configuración final depende de la configuración inicial de la red, la solución puede resultar ser un óptimo local, en lugar del óptimo global, consume mucho tiempo al analizar todas las variantes posibles así como calcular el flujo de carga de las redes radiales correspondientes.

1.4. 1.4 Patrón de Flujo Óptimo.

Si las impedancias de las ramas de la red se sustituyen por las resistencias de rama correspondientes, la distribución de flujo de carga que satisface a las leyes de Kirchhoff se llama Patrón de flujo óptimo. Cuando la distribución del flujo de carga en una rama, es un flujo optimo, las pérdidas de potencia de la red correspondiente, serán mínimas. Así, la idea básica del patrón de flujo óptimo es abrir el interruptor de la rama que tiene un valor mínimo de corriente en el lazo. [4]

Ventajas.

a) La configuración final de la red no dependerá de la topología de la red inicial.

b) La velocidad de cómputo es mucho más rápida que en el método de simple intercambio de rama.

c) El complicado problema de combinación de operación del interruptor se convierte en un problema heurístico debido a la apertura de un interruptor cada vez.

Desventajas (debido que todos los interruptores normalmente abiertos son cerrados en la red inicial)

a) Si hay muchos interruptores normalmente abiertos en una red, significa que el cálculo del flujo óptimo implica una gran cantidad de lazos. La solución final puede no ser óptima debido a los efectos mutuos entre lazos

b) Cuando el flujo de carga se resuelve mediante el método de la corriente de inyección equivalente, es necesario calcular la matriz de impedancia de la red equivalente de Thevenin con multipuertos. Esto aumentará la carga de cálculo.

c) Es necesario calcular el flujo de carga de los lazos de la red dos veces para cada operación de un solo interruptor (antes y después de abrir un interruptor).

1.5. Búsqueda Tabú.

La búsqueda Tabú, a diferencia de otros algoritmos basados en técnicas aleatorias de búsqueda de soluciones cercanas, se caracteriza porque utiliza una estrategia basada en el uso de memoria para escapar de los óptimos locales, en los que se puede caer al “moverse” de una solución a otra por el espacio de soluciones. Este algoritmo se dota, por tanto, de una “memoria” donde se almacenan los últimos movimientos realizados, y que puede ser utilizada para “recordar” aquellos movimientos que hacen caer de nuevo en soluciones ya exploradas, impidiendo así la evolución hacia ellas.[5]

1.6. Redes Neuronales.

Las redes neuronales son un método de resolución basado en el sistema nervioso biológico, este consiste en capas de unidades procesadoras, llamadas nodos,

unidas por conexiones direccionales: una capa de entrada, una capa de salida y cero o más capas ocultas en medio. Se le presenta un patrón inicial de entrada a la capa de entrada y luego los nodos que se estimulan transmiten una señal a los nodos de la siguiente capa a la que están conectados. Si la suma de todas las entradas que entran en una de estas neuronas es mayor que el famoso patrón de activación, esa neurona se activa, y transmite su propia señal a las neuronas de la siguiente capa. El patrón de activación, por tanto, se propaga hacia delante hasta que alcanza a la capa de salida, donde es devuelto como solución a la entrada presentada. Al igual que en el sistema nervioso de los organismos biológicos, las redes neuronales aprenden y afinan su rendimiento a lo largo del tiempo, mediante la repetición de rondas en las que se ajustan sus umbrales hasta que la salida real coincide con la salida deseada para cualquier entrada dada. [6]

1.7. Optimización de Colonias de Hormigas.

El algoritmo ACO (*Ant Colony Optimization*) es una técnica probabilística para solucionar problemas computacionales que pueden reducirse a buscar los mejores caminos o rutas en grafos, basado en el comportamiento de las hormigas cuando estas están buscando un camino entre la colonia y una fuente de alimentos, pues las mismas inicialmente vagan de manera aleatoria, al azar, y una vez encontrada comida regresan a su colonia dejando un rastro de feromonas. Si otras hormigas encuentran dicho rastro, es probable que estas no sigan caminando aleatoriamente, puede que estas sigan el rastro de feromonas, regresando y reforzándolo si estas encuentran comida finalmente. Sin embargo, al paso del tiempo el rastro de feromonas comienza a evaporarse, reduciéndose así su fuerza de atracción. Cuanto más tiempo le tome a una hormiga viajar por el camino y regresar de vuelta otra vez, más tiempo tienen las feromonas para evaporarse. Un camino corto, en comparación, es marcado más frecuentemente, y por lo tanto la densidad de feromonas se hace más grande en caminos cortos que en los largos. La evaporación de feromonas también tiene la ventaja de evitar convergencias a óptimos locales. Si no hubiese evaporación en absoluto, los caminos elegidos por

la primera hormiga tenderían a ser excesivamente atractivos para las siguientes hormigas. En este caso, el espacio de búsqueda de soluciones sería limitado. [7]
Un ejemplo de su aplicación, y más específicamente, de su aplicación en la ingeniería eléctrica es el expuesto en el artículo: Reconfiguración de sistemas de distribución de energía eléctrica usando un algoritmo de optimización basado en colonia de hormigas, de Sandra X. Carvajal, Jesús M. López, y Cesar A. Lemoine, en este se plantean una serie de pasos a seguir para la implementación de este algoritmo en nuestro tema de interés, y esos pasos serían los siguientes:
Paso 1: se define el número de hormigas de la colonia y se posicionan todas en los nodos iniciales de los alimentadores. Se establecen los valores iniciales para la intensidad de feromona de todos los ramos del sistema.
Paso 2: cada individuo selecciona un camino por seguir, teniendo en cuenta la intensidad de feromona y una función de mérito preestablecida, de acuerdo con las características físicas y topológicas de los sistemas de distribución.
Paso 3: después de construir las configuraciones, la función objetivo es evaluada con las restricciones del problema de reconfiguración (se debe calcular el flujo de carga), analizándose el desempeño de cada individuo separadamente.
Paso 4: se actualiza la intensidad de feromona. De la misma forma que acontece con las hormigas reales, la feromona virtual se evapora con el pasar del tiempo, perdiendo intensidad. Por otro lado, si una topología dada es establecida, la intensidad de la feromona se incrementa a través de una regla de transición.
Paso 5: se verifica la convergencia del algoritmo, finalizando el proceso si el número máximo de iteraciones es alcanzado, o cuando todas las hormigas seleccionen la misma configuración.

1.8. Optimización de Enjambre de Partículas.

El algoritmo de PSO (*Particle Swarm Optimization*) es una técnica metaheurística poblacional basada en la naturaleza, específicamente, en el comportamiento social del vuelo de las bandadas de aves, el movimiento de los bancos de peces, y de insectos, fue originalmente desarrollado en Estados Unidos por el sociólogo James Kennedy y por el ingeniero Russ C. Eberhart en 1995. En PSO los agentes

de búsqueda (partículas) intercambian información, las partículas modifican su dirección en función de las direcciones de las partículas de su vecindario, este almacena la experiencia propia o historia de cada agente, la partícula decide su nueva dirección en función de la mejor posición por la que pasó anteriormente, suele tener una convergencia rápida a buenas soluciones, la población del algoritmo se inicia de forma aleatoria y evoluciona iteración tras iteración, la búsqueda persigue siempre la solución más óptima posible, la búsqueda se basa exclusivamente en los valores de la función objetivo, trabaja con la información del problema codificada, es una técnica estocástica referida en fases (inicialización y transformación), tiene operadores de movimiento pero no de evolución como la mutación o el cruzamiento, no crea nuevas partículas durante su ejecución, sino que siempre son las mismas partículas iniciales modificadas a lo largo del proceso. [8]

1.9. Recocido Simulado (SA).

El algoritmo de recocido simulado (*Simulated Annealing*) está basado en la simulación de recocido de sólidos. Recocido denota un proceso de calentamiento de un sólido a una temperatura en la que sus granos deformados recristalizan para producir nuevos granos. En el proceso se comienza con una elevación de la temperatura, acto seguido viene un proceso de enfriamiento en donde la temperatura se baja poco a poco. De esta manera, cada vez que se baja la temperatura, las partículas se reacomodan en estados de más baja energía hasta que se obtiene un sólido con sus partículas acomodadas conforme a una estructura de cristal. Sin embargo, si el proceso de enfriamiento es demasiado rápido y no se alcanza en cada etapa el equilibrio térmico de las partículas, el sólido congelará en un estado cuya estructura será amorfa (está caracterizada por una imperfecta cristalización del sólido) en lugar de la estructura cristalina de más baja energía.

De esta manera se puede comparar el proceso de recocido simulado con un algoritmo de optimización, se tomarían a los estados de un sistema físico como

equivalencias de soluciones de un problema de optimización, y el valor de una solución sería la temperatura de un estado, en el algoritmo la solución va mutando (acercándose a la función de aptitud) y compara la aptitud anterior con la que se obtuvo y si es mayor conserva la nueva, en caso contrario el algoritmo toma la decisión de conservarla o no en base a la temperatura, si la temperatura es alta, como lo es al principio, pueden conservarse incluso cambios que causan decrementos significativos en la aptitud, y utilizarse como base para la ronda del algoritmo, por lo que se evita caer en mínimos locales, pero al ir disminuyendo la temperatura, el algoritmo se va haciendo más y más propenso a aceptar sólo los cambios que aumentan la aptitud convergiendo en un final en un mínimo. [9]

1.10. Algoritmos Genéticos.

En la naturaleza, los individuos de una especie compiten constantemente con otros por los recursos. Los individuos que tienen más éxito en la lucha por los recursos tienen mayores probabilidades de sobrevivir y generalmente una descendencia mayor. A diferencia de los menos adaptados que pueden no llegar a tener descendencia. Esto implica que los genes de los individuos mejor adaptados se propagarán a un número cada vez mayor de individuos de las sucesivas generaciones. La combinación de características buenas de diferentes ancestros puede originar, en ocasiones, que la descendencia esté incluso mejor adaptada al medio que los padres. De esta manera, las especies evolucionan adaptándose más y más al entorno a medida que transcurren las generaciones**).** Pero la adaptación de un individuo al medio no sólo está determinada por su composición genética. Influyen otros factores como el aprendizaje, en ocasiones adquirido por el método de prueba y error, en ocasiones adquirido por imitación del comportamiento de los padres**.** En general la evolución es la unión de la acción de la selección natural y la recombinación de material genético que ocurre durante la reproducción. De ahí que los Algoritmos Genéticos (AGs), de manera general, sean un método heurístico basado en la computación evolutiva que copia el proceso de evolución de las especies, comentado anteriormente, y expuesto por Darwin en su obra: *El Origen de las Especies por medio de la Selección Natural: la*

selección natural, la reproducción y la diversidad genética de individuos , este método crea una población que estaría compuesta por posibles soluciones a la problemática que se tendría en cuestión; de esta población tomaría los individuos de mayor capacidad de adaptación, que se llamarían padres, y el algoritmo se encargaría entonces de someter estos individuos a procesos de cruzamiento y mutación, para obtener nuevas soluciones, que se llamarían hijos, los cuales serían comparados con funciones de aptitud que serían las restricciones, y escogería solo los que fueran más aptos para formar otra población que debe ser mejor solución para la problemática; así se volvería a repetir los mismos procesos de forma iterativa hasta alcanzar una población compuesta de individuos que se acerquen lo más posible a la mejor solución. [10]

1.10.1 Orígenes de los Algoritmos Genéticos.

A principio de los años 60, Jhon Holland, investigador de la universidad de Michigan, asombrado por la capacidad de la naturaleza de perfeccionar a sus organismos, se interesó por los mecanismos de selección natural, y comenzó a desarrollar estas ideas y a adaptarlas para la resolución de problemas computacionales, centrándose en dos ideas fundamentales: imitar los procesos adaptativos de los sistemas naturales y diseñar sistemas informáticos capaces de resolver problemas usando los mecanismos más importantes presentes en la naturaleza. A la técnica que inventó Holland se le llamó originalmente "planes reproductivos", pero se hizo popular bajo el nombre "algoritmo genético" tras la publicación de su libro en 1975. Algunos años después de que Holland tuviera estas ideas conoció a un ingeniero industrial llamado David Golberg, que se convirtió en su estudiante y comenzó a aplicar los algoritmos genéticos a problemas industriales, logrando así con esta aplicación y otras que habían desarrollado en otros campos, darle la popularidad de la que tanto gozan hoy en día.

1.10.2 Como saber cuándo usar los Algoritmos Genéticos.

A pesar de demostrar que son muy buenos a la hora de resolver problemas de optimización, no siempre es factible el uso de estos, es por eso que se recomienda cumplir con los siguientes requerimientos antes de usarlos [11]:
1) Su espacio de búsqueda (sus posibles soluciones) debe estar delimitado dentro de un cierto rango.

2) Debe poderse definir una función de aptitud que nos indique qué tan buena o mala es una cierta respuesta.

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

1.10.3 Algunas Ventajas y Desventajas de los Algoritmos Genéticos.

Ventajas[6]**:**
1) Son intrínsecamente paralelos, es decir, operan de forma simultánea con varias soluciones, en vez de trabajar de forma secuencial como las técnicas tradicionales. Esto significa que la mayoría de los algoritmos son en serie, sólo pueden explorar el espacio de soluciones hacia una solución en una dirección al mismo tiempo, y si la solución que descubren resulta subóptima, no se puede hacer otra cosa que abandonar todo el trabajo hecho y empezar de nuevo. Mientras que los AGs sin llegan a una solución no válida la desechan y continúan con valores más prometedores.

2) Los AGs pueden trabajar con problemas no lineales, que son aquellos en los cuales el espacio de soluciones es muy grande. En un sentido general, la no linealidad es la norma, donde cambiar un componente puede tener efectos en cadena en todo el sistema, y donde cambios múltiples que, individualmente, son perjudiciales, en combinación pueden conducir hacia mejoras en la aptitud mucho mayores lo cual es lo más común en la vida real.

3) Tienen buen desenvolvimiento ante problemas con un paisaje adaptativo complejo (aquéllos en los que la función de aptitud es discontinua, ruidosa, cambia con el tiempo, o tiene muchos óptimos locales), las técnicas probabilísticas que usa le ayudan a evitar caer en óptimos locales (soluciones que son mejores que todas las que son similares a ella, pero que no son mejores que otras soluciones distintas situadas en algún otro lugar del espacio de soluciones).

4) Estos pueden manipular muchos parámetros simultáneamente, lo que significa que pueden trabajar con problemas multi-objetivos. Como son intrínsecamente paralelos pueden producir soluciones que optimicen parámetros distintos en una misma solución.

Desventajas:

1) El lenguaje para representar soluciones candidatas debe ser robusto, es decir, capaz de soportar cambios aleatorios. De ahí que se usen mayormente códigos binarios para representar a los individuos de la población.

2) Los parámetros de los AGs deben ser elegidos cuidadosamente para evitar errores, la función de aptitud debe ser bien definida, de lo contrario se obtendrían soluciones que no serían las deseadas para el problema que se tiene, si el tamaño de la población es muy pequeño el algoritmo puede no explorar lo suficiente para encontrar la mejor solución, etc.

3) En cuanto a la convergencia estos pueden converger prematuramente, debido a que si un individuo que es más apto que la mayoría de sus competidores emerge muy pronto en el curso de la ejecución, se puede reproducir tan abundantemente que merme la diversidad de la población demasiado pronto, provocando que el algoritmo converja hacia el óptimo local que representa ese individuo, en lugar de rastrear el paisaje adaptativo lo bastante a fondo para encontrar el óptimo global **,** en otros casos estos pueden tardar mucho en converger o no converger en absoluto, dependiendo en cierta medida de los parámetros que se utilicen, como el tamaño de la población, el número de generaciones, etc.

1.10.4 Algoritmo Genético Simple.

Se aplica a problemas de optimización en los que se parte de una población finita y bien definida, la función objetivo converge hacia una solución óptima y los parámetros a evaluar son de tipo continuo. Es el tipo de algoritmo más empleado y fácil de programar.

El Algoritmo Genético Simple, también denominado Canónico, se representa en el siguiente código. Como se verá a continuación, se necesita una codificación o representación del problema, que resulte adecuada al mismo. Además se requiere una función de ajuste o adaptación al problema, la cual asigna un número real a cada posible solución codificada. Durante la ejecución del algoritmo, los padres deben ser seleccionados para la reproducción, a continuación dichos padres seleccionados se cruzarán generando dos hijos, sobre cada uno de los cuales actuará un operador de mutación. El resultado de la combinación de las anteriores funciones será un conjunto de individuos (posibles soluciones al problema), los cuales en la evolución del Algoritmo Genético formarán parte de la siguiente población.[12]

BEGIN/* Algoritmo Genético Simple */
Generar una población inicial.
Computar la función de evaluación de cada individuo.
WHILE NOT Terminado DO
BEGIN /* Producir nueva generación */
FOR Tamaño población/2 DO
BEGIN /*Ciclo Reproductivo */
Seleccionar dos individuos de lo anterior generación,
para el cruce (probabilidad de selección proporcional
a la función de evaluación del individuo).
Cruzar con cierta probabilidad los dos
Individuos obteniendo dos descendientes.
Mutar los dos descendientes con cierta probabilidad.
Computar la función de evaluación de los dos

descendientes mutados.
Insertar los dos descendientes mutados en la nueva generación.
END
IF la población ha convergido THEN
Terminado := TRUE
END
END

1.10.5 Algoritmos Genéticos Paralelos (AGP).

Normalmente se utiliza en ámbitos en los que se precisa localizar múltiples soluciones derivadas de la aplicación concurrente de varias funciones objetivo. El resultado es que la optimización no converge hacia un máximo global, sino hacia varias soluciones igualmente válidas. Estos algoritmos funcionan evaluando y manteniendo subespecies (o "nichos de población") que evolucionan con criterios diferentes y compiten entre sí.

Surgen ante la necesidad de computo requerida por problemas de extrema complejidad, cuyo tiempo de ejecución utilizando los tradicionales algoritmos genéticos secuenciales es prohibitivo. Es por eso que se buscó la manera de poder adaptar este tipo de heurísticas a distintas configuraciones de cómputo paralelo, lo que dio lugar a tres grandes modelos de algoritmos genéticos paralelos: (1) algoritmos maestro-esclavo, (2) algoritmos de grano fino y (3) algoritmos de grano grueso.[13, 14]

1.10.6 Algoritmos maestro-esclavo:

Estos algoritmos trabajan con una única población de individuos que será gestionada por el nodo maestro. La evaluación de la adecuación de los individuos y/o la aplicación de los operadores genéticos puede ser realizada por los nodos esclavos. A cada nodo esclavo le corresponderá una parte de la población total, sobre la cual realizará las operaciones antes citadas. Una vez terminado este proceso, devolverán el resultado al nodo maestro, que realizará la selección de individuos. Normalmente, la operación que se suele implementar en paralelo es la evaluación de la adecuación de los individuos, porque suele ser la más compleja.

Además, este valor es independiente del resto de la población, por lo que su implementación es muy sencilla.

El intercambio de información entre los nodos es sencillo: el nodo maestro envía el subconjunto de individuos que corresponde a cada nodo, y estos le devuelven los valores de adecuación para cada uno de ellos.[15]

2) Algoritmos de grano fino:

Este tipo de algoritmos han sido diseñados para ser implementados usando computadores masivamente paralelos. Ahora, la población se encuentra dividida espacialmente entre los distintos procesadores e, idealmente, cada procesador debería albergar un único individuo. El cruce y la selección de individuos se harán entre individuos que pertenezcan a un mismo vecindario, formado por un conjunto de individuos adyacentes. Sin embargo, se permite el solapamiento entre vecindarios para propiciar la interacción, aunque leve, entre todos los individuos de la población.[16]

3) Algoritmos de grano grueso:

Las características más importantes de estos algoritmos son el uso de múltiples poblaciones y la migración de individuos entre ellas. Dado que cada una de las poblaciones evoluciona independientemente, el ratio de migración será muy importante de cara a obtener resultados satisfactorios. Hay que aclarar que existen algunos factores que son determinantes a la hora de escoger que tipo de algoritmo seleccionar, si los tradicionales algoritmos genéticos secuenciales o los algoritmos genéticos paralelos; entre los más importantes están la frecuencia con que se realicen las migraciones y la topología de comunicación entre las subpoblaciones. En el primer caso este es un aspecto muy importante, ya que una migración prematura de individuos puede entrañar serios problemas, ya que la calidad de los individuos enviados podría no aportar ninguna mejora en el resto de poblaciones, con lo que se estaría desperdiciando valiosos y costosos recursos de red. Sin embargo, una migración muy tardía puede afectar también negativamente y aumentar considerablemente el tiempo de convergencia al óptimo global o, incluso, propiciar que esta no se produzca. Para el segundo caso si el grado de conectividad es alto o el diámetro de la red es pequeño (o ambas cosas) las

soluciones buenas se expandirán rápidamente a través de todas las poblaciones, favoreciendo así la evolución hacia el óptimo. En caso contrario, las poblaciones estarán más aisladas las unas de las otras, por lo que evolucionaran mucho más despacio, y llevará más tiempo la incorporación de los genes potencialmente buenos a las nuevas soluciones. Por otro lado, es necesario tener en cuenta que la densidad de conexiones puede influir negativamente en el algoritmo, ya que supone una sobrecarga importante en el tráfico de la red. [17]

1.10.7 Comparación de los Algoritmos Genéticos con otras técnicas heurísticas.

La amplia gama de problemas resueltos exitosamente en diversas áreas han popularizado a esta técnica evolutiva como una de las más versátiles y robustas. Los algoritmos genéticos son muy útiles para resolver problemas no lineales (problemas con espacio de soluciones de dimensión elevada), problemas no muy bien comprendidos de antemano, también en donde el diseño de operadores específicos no sea sencillo, en problemas multiobjetivos donde los métodos heurísticos tradicionales pueden quedar atrapados en óptimos locales, o en casos donde no es necesario obtener una solución óptima al problema sino que una buena solución aproximada sería suficiente. Los algoritmos genéticos presentan ciertas características que los hacen ventajosos cuando se los compara con otras técnicas de resolución de problemas. A pesar que necesita trabajar con lenguajes robustos para representar las posibles soluciones, como lo son los números binarios, este solo necesita una buena definición de su función de aptitud y solo eso, no necesita más datos del problema para obtener una solución. La independencia del mecanismo de búsqueda evolutivo de las características del problema permite a los algoritmos genéticos evitar, en ocasiones, el estancamiento en mínimos locales del problema en los cuales son proclives a caer ciertos algoritmos tradicionales basados en gradientes u otras búsquedas locales dirigidas**.** De ahí que se considere a los algoritmos genéticos como un lenguaje de programación robusto y versátil.

CAPÍTULO 2. MATERIALES Y MÉTODOS

La creación de la población es una cuestión esencial en la resolución de problemas por algoritmos genéticos, ya que sin la variabilidad genética suficiente no se obtienen las respuestas más óptimas requeridas. De ahí que los temas que se hablaran en este capítulo serán el algoritmo para la creación de la población, los diferentes métodos de creación de la población y la optimización del número de individuos.

1.11. Algoritmo de creación de la población:

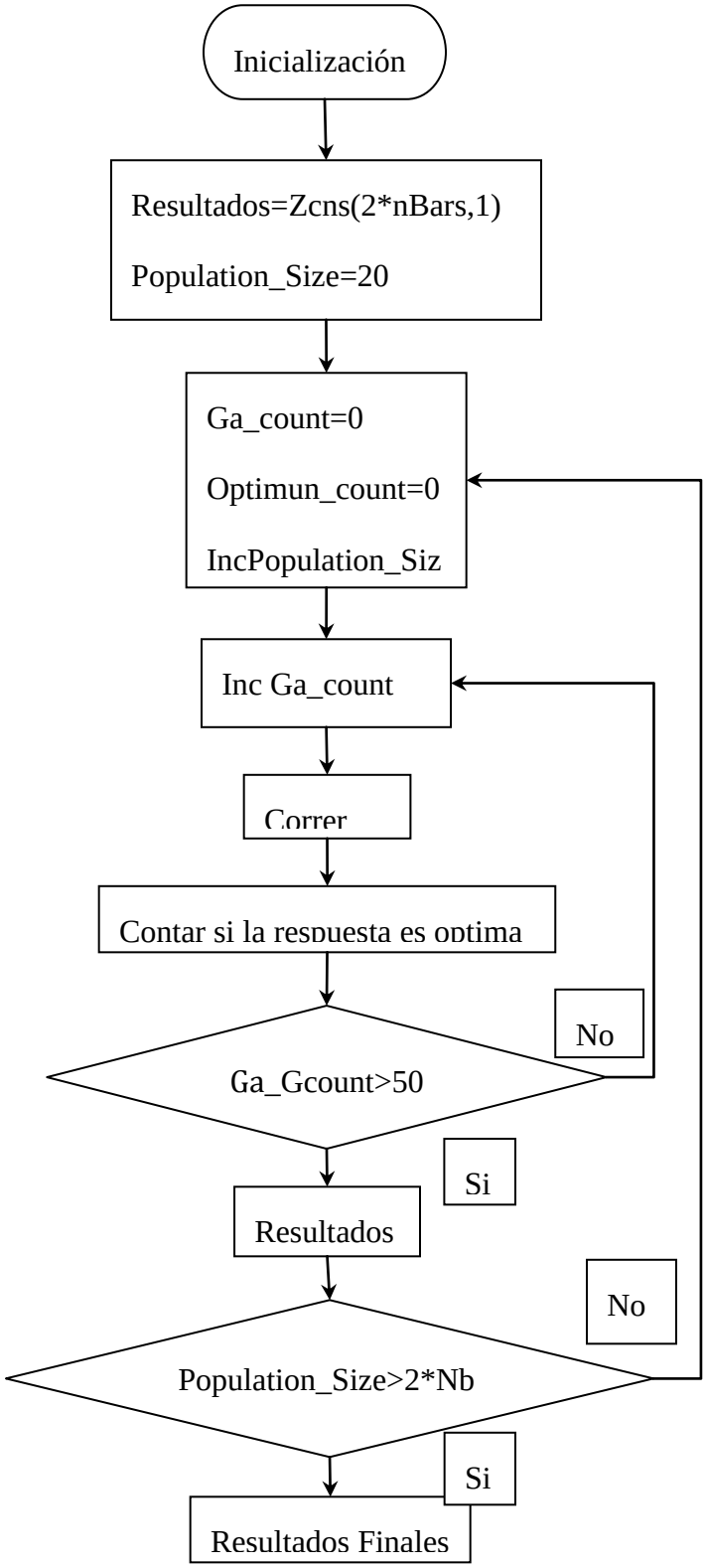


Figura 1: Algoritmo de creación de la poblacion

Se inicializa la función, los parámetros establecidos son una cuenta inicial de 20 individuos y como resultados los óptimos encontrados en dos veces el numero de barras (en este caso 30 barras).Se coloca una función Ga_count=0 y una función Optimun_count=0 para contar los óptimos en cada incremento del número de individuos que se lograra implementando la función inc population_Size. Luego se aplica la función inc Ga_count para que se aplique el incremento de los individuos al final de cada ronda de pruebas y se corre el Ga. Con el código se cuenta el número de óptimos al que se llego en cada ronda de pruebas. En este lazo hay un código que pregunta si Ga_cont<50, si la respuesta es no se regresa al evento posterior de la función inc Population_Size, si la respuesta es sí se pasa a contabilizar los resultados parciales que consiste en registrar el numero de óptimos alcanzados y el número de individuos para el que se esteban corriendo las pruebas. Luego se establece otro cuadro de preguntas en el que se pregunta si la population_Size>2*nBars. Si la respuesta es no se regresa a la cuenta de los individuos si la respuesta es sí se plotea una gráfica en la que se interceptan el número de individuos contra el número de éxitos alcanzados.

1.12. Creación de la Población Inicial

La creación de la población inicial es un paso importante dentro del proceso de ejecución de los algoritmos genéticos. Para el caso de la reconfiguración mediante algoritmos genéticos existen varias formas de obtener esta población inicial. .

Variantes de generación de la población inicial:

1. Generar aleatoriamente cada circuito o individuo de la población inicial.
2. Generar un circuito (individuo) de la población inicial y luego obtener los restantes mediante operaciones de mutación.
3. Generar un grupo que contenga información genética variada con todos los posibles interruptores abiertos.

1.12.1 Generar aleatoriamente cada circuito o individuo de la población inicial.

Esta variante genera los individuos uno a uno mediante la creación de un circuito radial partiendo desde cero. Para ello se emplea el Algoritmo de Prim.

La desventaja fundamental de esta forma de generar la población inicial radica en que el proceso de formación de cada nuevo individuo es aleatorio. Al final puede que no se obtenga toda la diversidad genética necesaria para obtener el individuo óptimo por la no presencia de dentro de la población de los genes que lo conforman como el individuo más apto. Es decir, dentro de todos los individuos de la población inicial no se encuentran todos genes del individuo más apto. La obtención de estos genes faltantes quedaría entonces a merced de los procesos de mutación, los cuales son más lentos que los procesos de cruzamiento.

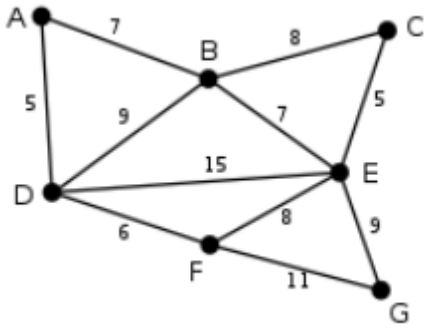
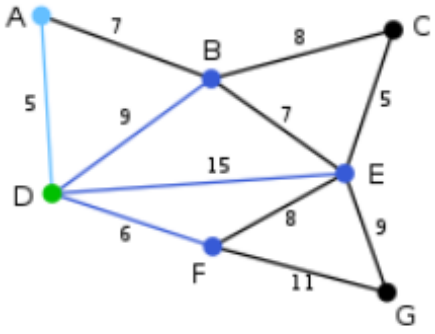
Otra desventaja es que este proceso de obtención de los individuos es más lento que otras variantes aleatorias de obtención de los individuos.

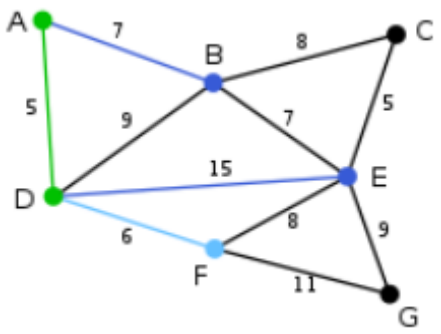
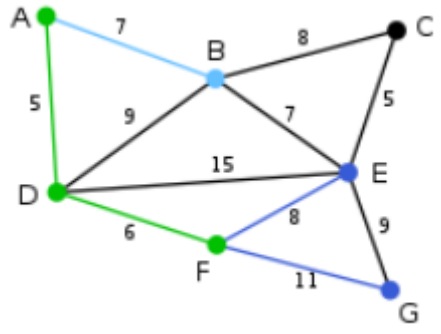
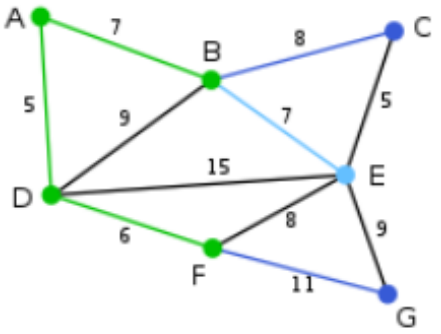
1.12.2 Algoritmo de Prim

El **algoritmo de Prim** es un algoritmo perteneciente a la teoría de los grafos para encontrar un árbol recubridor mínimo en un grafo conexo, **no** dirigido y cuyas aristas están etiquetadas.

En otras palabras, el algoritmo encuentra un subconjunto de aristas que forman un árbol con todos los vértices, donde el peso total de todas las aristas en el árbol es el mínimo posible. Si el grafo no es conexo, entonces el algoritmo encontrará el árbol recubridor mínimo para uno de los componentes conexos que forman dicho grafo no conexo.

El algoritmo fue diseñado en 1930 por el matemático Vojtech Jarnik y luego de manera independiente por el científico computacional Robert C. Prim en 1957 y redescubierto por Dijkstra en 1959. Por esta razón, el algoritmo es también conocido como **algoritmo DJP** o **algoritmo de Jarnik**. [18-26]

Image	Descripción	No visto	En el grafo	En el árbol
	Este es el grafo ponderado de partida. No es un árbol, ya que para serlo se requiere que no haya ciclos, y en este caso sí hay. Los números cerca de las aristas indican el peso. Ninguna de las aristas está marcada, y el vértice **D** ha sido elegido arbitrariamente como el punto de partida.	C, G	A, B, E, F	D
	El segundo vértice es el más cercano a **D**: **A** está a 5 de distancia, **B** a 9, **E** a 15 y **F** a 6. De estos, 5 es el valor más pequeño, así que marcamos la arista **DA**.	C, G	B, E, F	A, D

	El próximo vértice a elegir es el más cercano a **D** o **A**. **B** está a 9 de distancia de **D** y a 7 de **A**, **E** está a 15, y **F** está a 6. 6 es el valor más pequeño, así que marcamos el vértice **F** y a la arista **DF**.	C	B, E, G	A, D, F
	El algoritmo continua. El vértice **B**, que está a una distancia de 7 de **A**, es el siguiente marcado. En este punto la arista **DB** es marcada en rojo porque sus dos extremos ya están en el árbol y por lo tanto no podrá ser utilizado.	null	C, E, G	A, D, F, B
	Aquí hay que elegir entre **C**, **E** y **G**. **C** está a 8 de distancia de **B**, **E** está a 7 de distancia de **B**, y **G** está a 11 de distancia de **F**. **E** está más cerca, entonces marcamos el vértice **E** y la arista **EB**. Otras dos	null	C, G	A, D, F, B, E

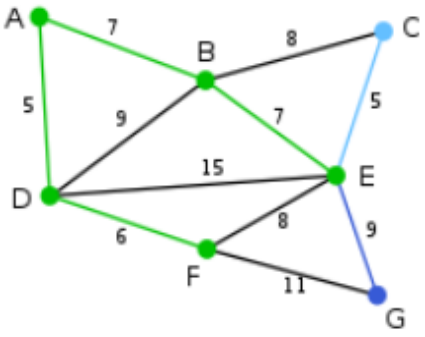
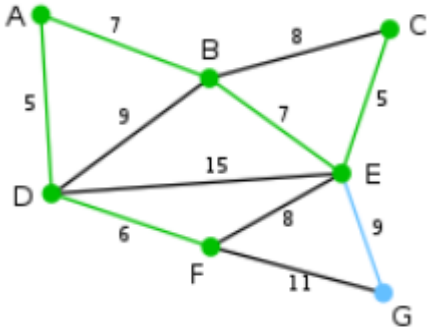
	aristas fueron marcadas en rojo porque ambos vértices que unen fueron agregados al árbol.			
	Sólo quedan disponibles **C** y **G**. **C** está a 5 de distancia de **E**, y **G** a 9 de distancia de **E**. Se elige **C**, y se marca con el arco **EC**. El arco **BC** también se marca con rojo.	null	G	A, D, F, B, E, C
	**G** es el único vértice pendiente, y está más cerca de **E** que de **F**, así que se agrega **EG** al árbol. Todos los vértices están ya marcados, el árbol de expansión mínimo se muestra en verde. En este caso con un peso de 39.	null	null	A, D, F, B, E, C, G

Figura 2: Ejemplo de ejecución de Algoritmo de Prim

1.12.3 Codigo de implementación del algoritmo por el método de Generar aleatoriamente cada individuo de la población inicial:

El algoritmo se implementa con el siguiente código:

```
totalPopulation = sum(options.PopulationSize);
initPopProvided = size(options.InitialPopulation,1);
individualsToCreate = totalPopulation - initPopProvided;
% Inicializar la población a crear
Population = true(totalPopulation,GenomeLength);
% Emplear la población inicial brindada
if initPopProvided > 0
    Population(1:initPopProvided,:) = options.InitialPopulation;
end
% Crear la población restante
linedata = ga_mpc.branch; busXY = []; HS = [];
rN = find(ga_mpc.bus(:,[2])==3);
for i = 1:individualsToCreate
   [L, RadialNet] = radialnetV3(linedata,rN,busXY,HS);
   setOFF = setdiff([1:GenomeLength],L); % Obtener las posiciones '0'
   Population(initPopProvided+i,setOFF) = 0;%Hacer '0'-> Lineas en OFF
end;
```

1.13. Creación de la población inicial mediante operaciones de mutación

Esta variante primero crea un individuo o circuito radial completamente desde cero. Posteriormente se toma este individuo ya creado y se le efectúa una

mutación. De esta operación surge un nuevo individuo que resulta diferente al primero. El proceso se repite hasta obtener todos los integrantes de la población inicial.

Computacionalmente esta variante es más rápida que la variante anterior, pues no es necesario comenzar el proceso completamente desde cero. Solamente se cambia la información genética del individuo en uno o varios de los puntos del circuito.

Este proceso no garantiza de por si la diversidad genética de la población inicial. La aleatoriedad del proceso de mutación puede no dar lugar a la obtención de todos los genes del individuo óptimo dentro de la sumatoria de toda la información genética obtenida.

1.13.1 Codigo de implementación del algoritmo por el método de Creación de la población inicial mediante operaciones de mutación:

El algoritmo se implementa con el siguiente código:

```
totalPopulation = sum(options.PopulationSize);
initPopProvided = size(options.InitialPopulation,1);
individualsToCreate = totalPopulation - initPopProvided;
mutationRate = 0.1; % Tasa de mutación
Population = true(totalPopulation,GenomeLength); % Inicializar
if initPopProvided > 0% Emplear la población brindada
   Population(1:initPopProvided,:) = options.InitialPopulation;
end
linedata = ga_mpc.branch; busXY = []; HS = [];
rN = find(ga_mpc.bus(:,[2])==3);
% Obtener un individuo inicial
first_ind  = []; % Crear desde la nada (conjunto vacio)
```

```
[L, RadialNet] = radialnetV3(linedata,rN,busXY,HS);
setOFF = setdiff([1:GenomeLength],L); % Obtener las posiciones '0'
Population(1,setOFF) = 0; % Hacer '0' -> Lineas en OFF
first_ind = Population(1,:);
Co_Tree_first_ind = find(first_ind==0);
for i = 1:individualsToCreate% Crear la población restante
   mutationPoints = find(rand(1,length(Co_Tree_first_ind)) < mutationRate);
   Co_Tree_Child  = Co_Tree_first_ind;
   child          = first_ind;
   for ii=1:length(mutationPoints)
      Closed_Branch = Co_Tree_Child(mutationPoints(ii));
      branch_data = ga_mpc.branch;
      branch_data(:,11) = 1; % First Close All lines
      branch_data(Co_Tree_Child,11) = 0;%Open lines of Co-Tree Child
      Radial_Net = child; % Child to mutate % first_ind
      [B] = GetBranchsInLoop(Closed_Branch, Radial_Net, branch_data);
      loop_branch = B;
      open_branch = loop_branch(ceil(length(loop_branch).*rand(1,1)));
      Co_Tree_Child(mutationPoints(ii)) = open_branch;
      child(Closed_Branch) = 1; % Toggle state
      child(open_branch)   = 0; % Toggle state
   end; % for ii
   mutationChildren(i,:) = child;
   Population(initPopProvided+i,:) = child;
```

```
end %for i
```

1.14. Generar una población genéticamente variada.

El objetivo de esta variante es generar un grupo que contenga información genética variada con todos los posibles interruptores abiertos. Es decir, al final quedarían dentro de los individuos conformados, todos los fragmentos genéticos del individuo óptimo. Esto no quiere decir que el individuo más apto esté dentro de los creados. En realidad, lo que se puede hacer es conformar el más apto mediante el adecuado intercambio genético (o cruzamiento) de esta población.

Los procesos de cruzamiento pueden no garantizar completamente la obtención del individuo mejor adaptado. Pero se está en mejores condiciones de lograrlo partiendo de una población inicial genéticamente variada, que cuando se parte de una población con menos variación.

1.15. Algoritmo de Creación de la Población Inicial

La población inicial se puede crear siguiendo el algoritmo siguiente:

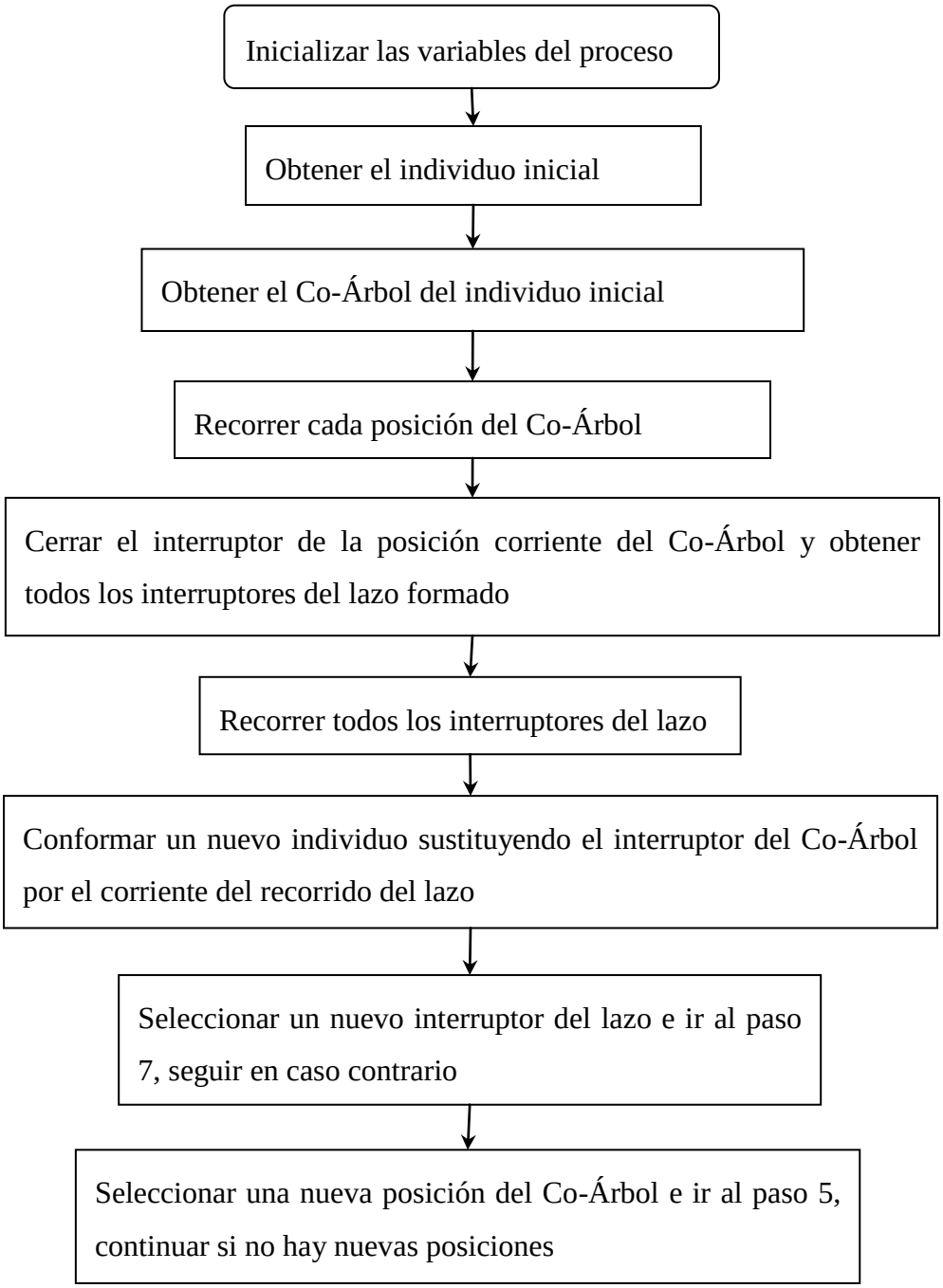


Figura 3:Algoritmo de creación de la población inicial.

Este algoritmo fue implementado en una función de MATLAB con los comandos mostrados a continuación:

```
% Inicializar la población a crear
    Population = true(totalPopulation,GenomeLength);
    % Emplear la población inicial brindada
    if initPopProvided > 0
        Population (1:initPopProvided,:) = options.InitialPopulation;
    end
    % Crear la población restante
    linedata = ga_mpc.branch; busXY = []; HS = [];
    rN = find(ga_mpc.bus(:,[2])==3);
    % Obtener un indivuduo inicial
    first_ind  = []; % Crear desde la nada (conjunto vacio)
    [L, RadialNet] = radialnetV3(linedata,rN,busXY,HS);
    setOFF = setdiff([1:GenomeLength],L); % Obtener las posiciones '0'
    Population(1,setOFF) = 0; % Hacer '0' -> Lineas en OFF
    first_ind = Population(1,:);
    % Conformar la poblacion
    Final_Group = [];
    Co_Tree_first_ind = find(first_ind==0);
    sP_Zs = zeros(1,length(Co_Tree_first_ind));
    for ii=1:length(Co_Tree_first_ind)
        Closed_Branch = Co_Tree_first_ind(ii);
        branch_data = ga_mpc.branch;
        branch_data(:,11) = 1; % Primero Cerrar Todas las Lineas
```

```
        branch_data(Co_Tree_first_ind,11) = 0; % Cerrar lineas
        Radial_Net = first_ind; % Individuo a mutar
        [B] = GetBranchsInLoop(Closed_Branch, Radial_Net, branch_data);
        New_Group = true(length(B)+1,GenomeLength);
        New_Group(1,:) = first_ind;
        Co_Tree_Array = [];
        sP_Zs(ii) = length(B)+1;
        for iii=1:length(B) % Viajar por todas las ramas en el laso
            New_Co_Tree = Co_Tree_first_ind;
            New_Co_Tree(ii) = B(iii); % Cambiar solo la rama corriente
            Co_Tree_Array = [Co_Tree_Array; New_Co_Tree];
            New_Group(iii+1,New_Co_Tree) = 0;
        end
        Final_Group = [Final_Group; New_Group];
    end
    Population = Final_Group;
```

Al final del algoritmo anterior deben quedar todos los individuos conformados dentro de la variable **Population**.

1.16. 2.3Optimizacion de individuos de la población:

En este epígrafe se muestran varias pruebas efectuadas al circuito de la IEEE de 30 barras mostrado sin en el anexo 1para optimizar el número de individuos con el objetivo de minimizar el tiempo que demora correr el algoritmo genético el número de veces necesario.

Para esto se utilizó el código siguiente:

```
    Success_Results = zeros(2*nBars,1);
    bStop_par = 1; PopulationSize_i = 20;
    while bStop_par,
        PopulationSize_i = PopulationSize_i + 1;
        options                                                    =
gaoptimset(options,'PopulationSize',PopulationSize_i,'Display','off');
        % Generations_i = Generations_i + 1;
        % options = gaoptimset(options,'Generations',Generations_i);
        bStop_ga = 1;
        ga_count = 0; Optimun_count = 0;
        while bStop_ga,
            ga_count = ga_count + 1;
            [x fval exitflag output population scores] = ...

ga(ObjectiveFunction,nvars,[],[],[],[],[],[],[],options);
            bStop_ga = ga_count < 50;
            % Optimun_Reached = all(x == x_best_case30); % x_my_case14
x_best_case30
            Optimun_Reached = fval < 3.0;
            Optimun_count = Optimun_count + Optimun_Reached;
            % display(['Optimun Reached = ' num2str(Optimun_Reached)]);
        end;
        display('----------------------------------------');
        display(['Total of success = ' num2str(Optimun_count) ' ('
num2str(PopulationSize_i) ')']);
        display('----------------------------------------');


        Success_Results(PopulationSize_i) = Optimun_count;
        bStop_par = PopulationSize_i <2*nBars;
    end;
    figure; plot(Success_Results);
end;
```

En donde se empieza la prueba a partir de 20 individuos efectuando al principio con 25 pruebas y a partir de ahí ascendiendo hasta llegar a 50 pruebas.

Ejemplo 1:

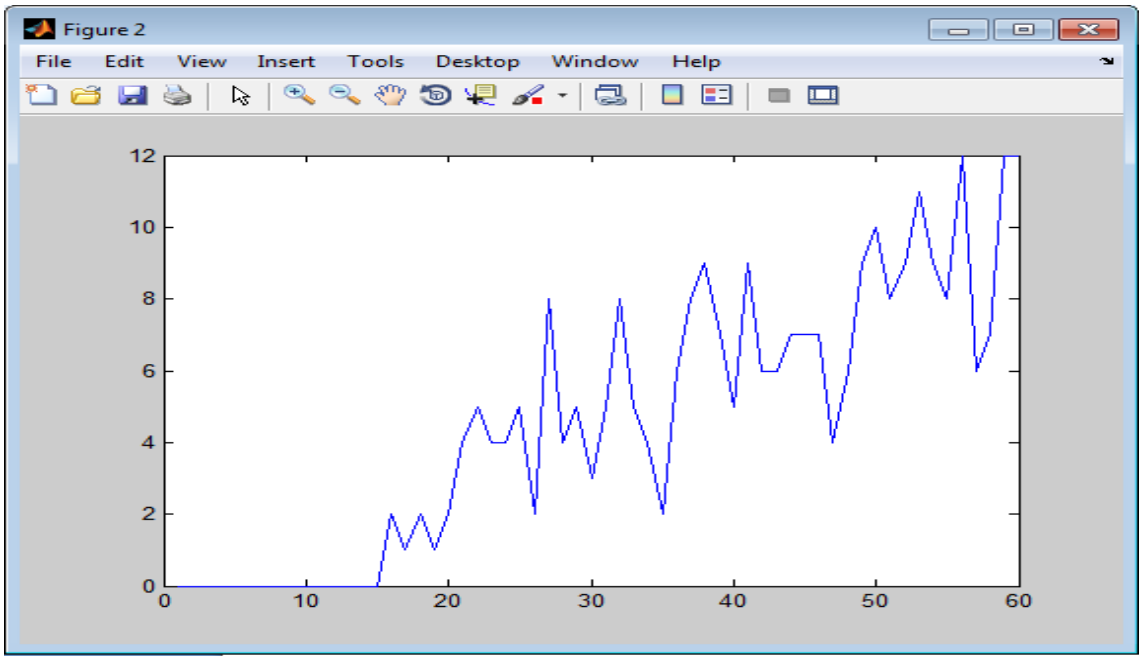


Figura 2.31: Corrida a partir de 15 individuos hasta 60 y margen de éxitos menor de 3.0 (20 pruebas)

Ejemplo 2:

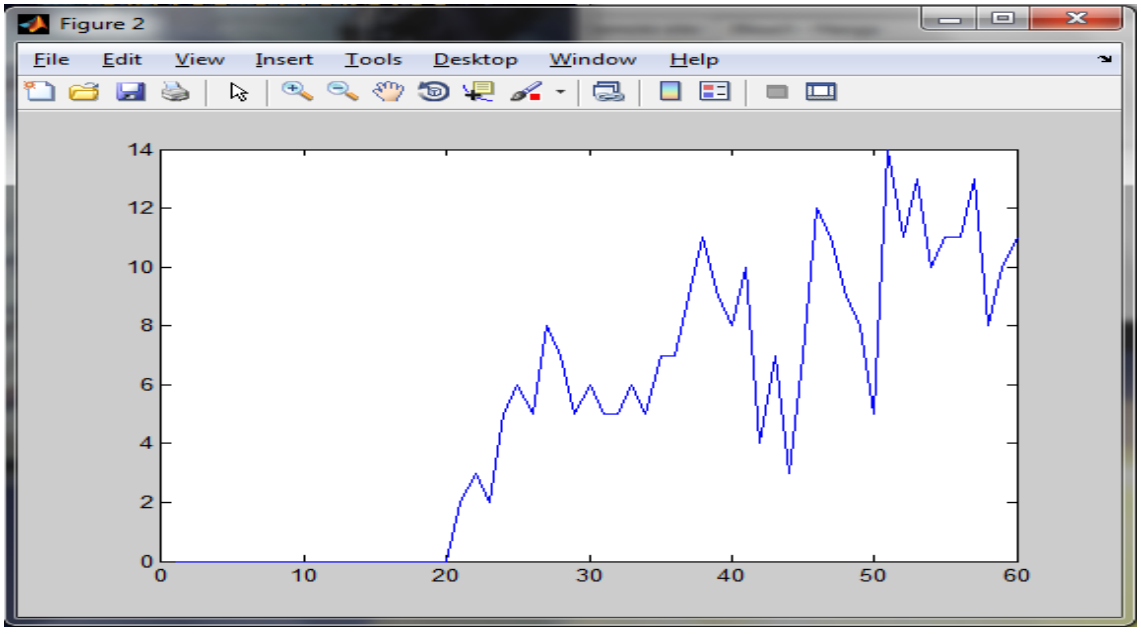


Figura 2.32: Corrida a partir de 15 individuos hasta 60 y margen de éxitos menor de 3.0 (25 pruebas)

Ejemplo 3:

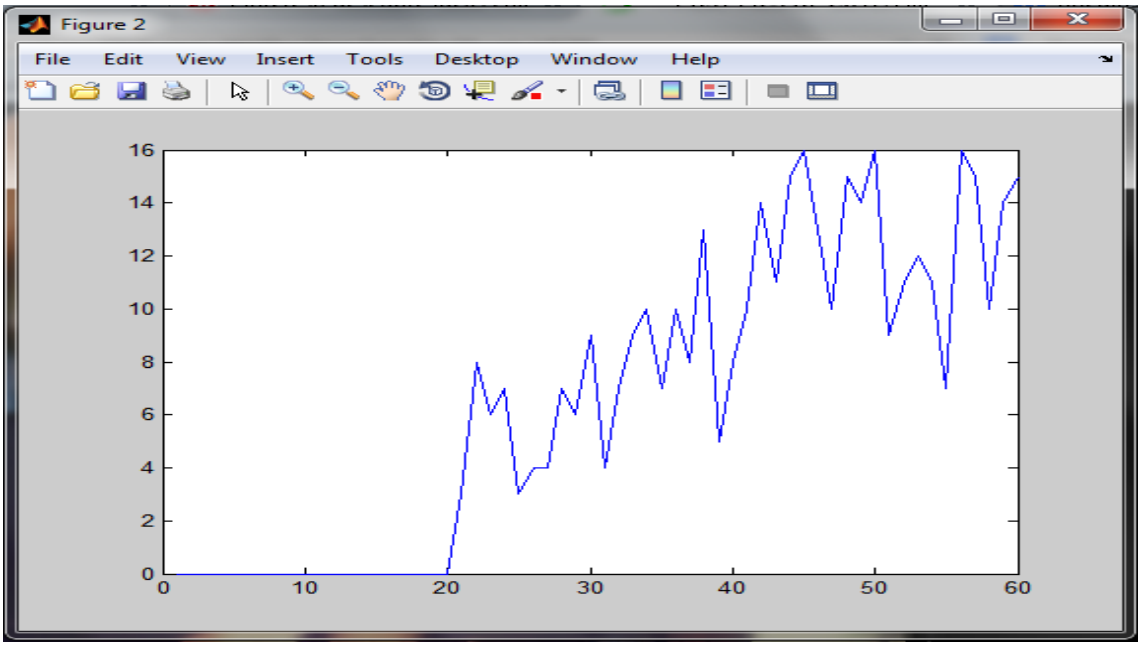


Figura 2.33: Corrida a partir de 15 individuos hasta 60 y margen de éxitos menor de 3.0 (30 pruebas)

Ejemplo 4:

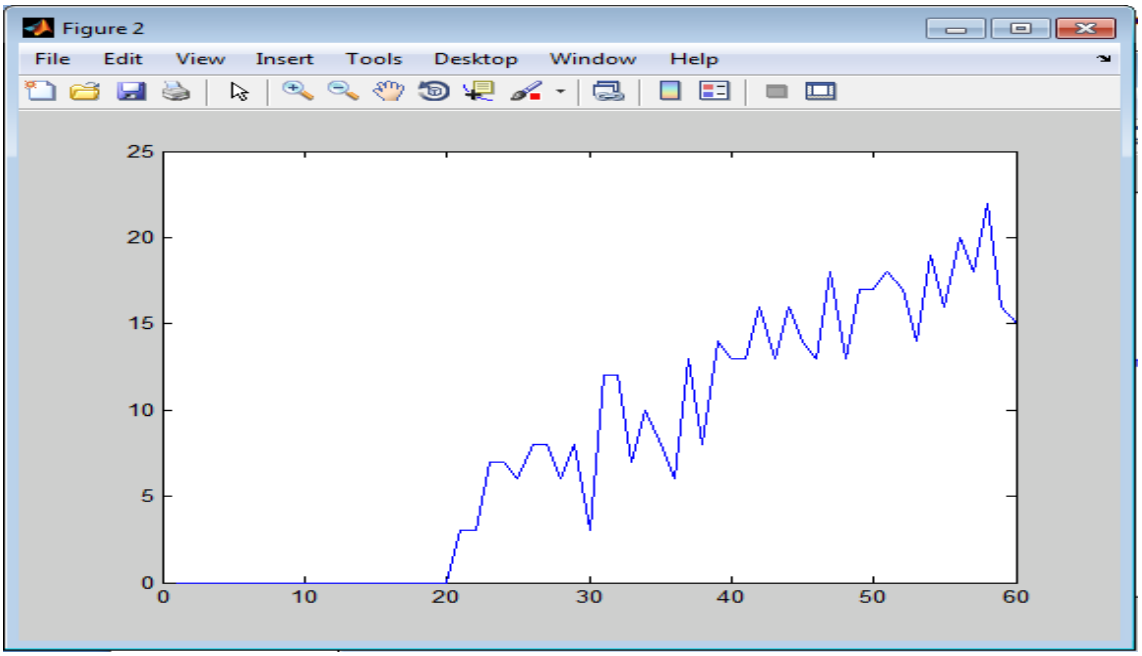


Figura 2.34: Corrida a partir de 15 individuos hasta 60 y margen de éxitos menor de 3.0 (35 pruebas)

Ejemplo 5:

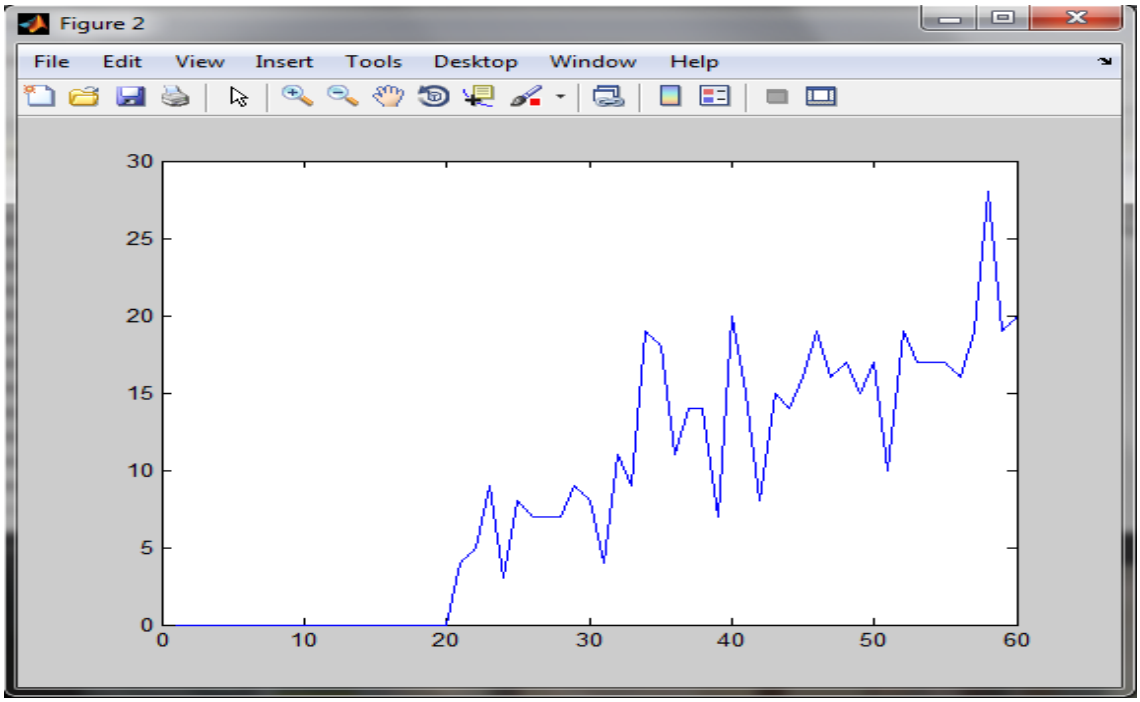


Figura 2.35: Corrida a partir de 20 individuos hasta 60 y margen de éxitos menor de 3.0 (40 pruebas)

Ejemplo 6:

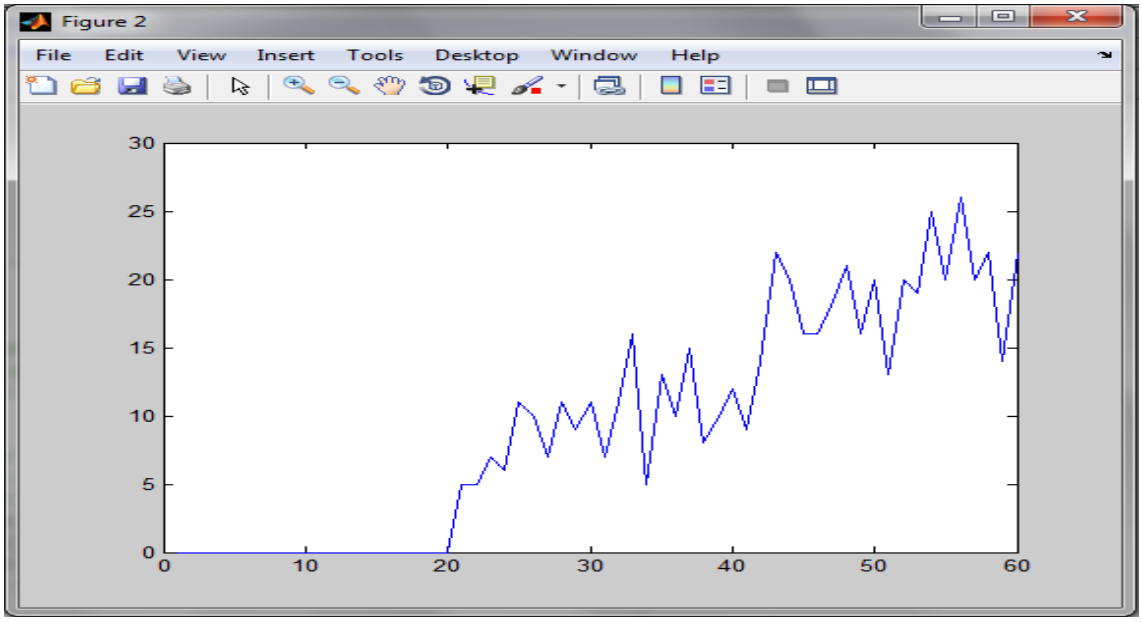


Figura 2.36: Corrida a partir de 20 individuos hasta 60 y margen de éxitos menor de 3.0 (45 pruebas)

Ejemplo 7:

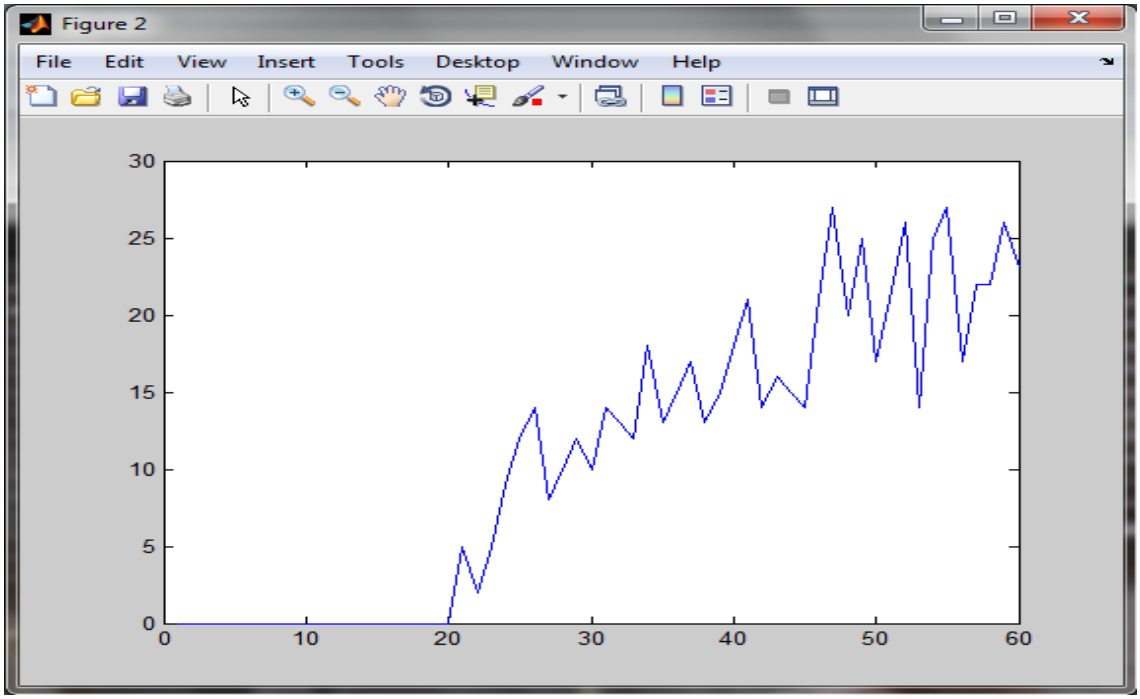


Figura 2.37: Corrida a partir de 20 individuos hasta 60 y margen de éxitos menor de 3.0 (50 pruebas)

CAPÍTULO 3. RESULTADOS Y ANALISIS DE RESULTADOS

En este capítulo se tratara la rapidez con que se crea la población inicial y se establecerá una comparación entre los tres métodos de creación de población expuestos en el capítulo anterior, también se analizaran algunos casos concretos tomando como base circuitos establecidos por la IEEE de 118, 57 y 30 barras para ilustrar la rapidez con que se crea la población en circuitos de diferentes grados de complejidad.

1.1. Códigos empleados en la comparación de métodos:

Este es el fragmento de código empleado para medir los tiempos comparando los métodos de creación de individuos de la población entre ellos:

```
% Poblacion Inicial
    options = gaoptimset(options,'Generations',1200);
    % Metodo 1: aleatoria desde cero uno a uno
    options                                                    =
gaoptimset(options,'CreationFcn',@reconfig_gacreationuniform);
    tic;
    [x fval exitflag output population scores] = ...

ga(ObjectiveFunction,nvars,[],[],[],[],[],[],[],options);
    t1 = toc,
    % Metodo 2: aleatoria desde un individuo
    options = gaoptimset(options,'CreationFcn',@reconfig_gadiverse);
    tic;
    [x fval exitflag output population scores] = ...

ga(ObjectiveFunction,nvars,[],[],[],[],[],[],[],options);
    t2 = toc,
```

```
    % Metodo 3: aleatoria desde un individuo por mutacion
    options                                                    =
gaoptimset(options,'CreationFcn',@reconfig_ga_new_by_mutation);
    tic;
    [x fval exitflag output population scores] = ...

ga(ObjectiveFunction,nvars,[],[],[],[],[],[],[],options);
    t3 = toc,
```

1.2. Ejemplos de casos concretos

En los siguientes ejemplos se hacen varias corridas del algoritmo genético y se mide el tiempo de creación de población para analizar con los parámetros ya modificados, cuál de los métodos es más rápido y cual ofrece mayor diversidad genética para mejor obtención de resultados óptimos:

1.2.1 Para circuito de 30 barras:

Ejemplo 1:

ga_mpc =

version: '2'

baseMVA: 100

bus: [30x13 double]

gen: [6x21 double]

branch: [41x13 double]

areas: [3x2 double]

gencost: [6x7 double]

Elapsed time is 1.592038 seconds.

Generation	Best f-count	Best f(x)	Mean f(x)	Stall Generations
1	60	3.7	6.374e+005	0
2	90	3.7	340.2	1
3	120	3.7	5.166	2

4	150	3.7	5.583	0
5	180	3.233	5.364	0
6	210	3.233	4.549	1
7	240	3.233	4.201	0
8	270	3.233	4.187	1
9	300	3.218	3.673	0
10	330	3.021	3.786	0
11	360	3.021	3.687	1
12	390	3.021	4.064	2
13	420	3.021	3.622	3
14	450	3.021	3.196	4
15	480	3.021	3.288	5
16	510	3.021	3.062	6
17	540	3.021	3.152	7
18	570	3.007	3.152	0
19	600	3.007	3.051	1
20	630	3.007	3.07	2
21	660	3.007	3.116	3
22	690	3.007	3.028	4
23	720	3.007	3.057	5
24	750	3.007	3.161	6
25	780	2.96	3.368	0
26	810	2.96	3.298	1
27	840	2.96	3.411	2
28	870	2.96	3.437	3
29	900	2.96	3.251	4
30	930	2.96	3.151	5

t1 =

6.3555

Elapsed time is 0.070412 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	102	4.47	2148	0
2	132	4.47	1260	1
3	162	4.47	960.4	2
4	192	4.345	959.2	0
5	222	4.345	1274	1
6	252	4.217	942	0
7	282	3.72	442	0
8	312	3.382	441.8	0
9	342	3.382	441.5	1
10	372	3.382	441.3	2
11	402	3.382	4.098	3
12	432	3.382	3.992	4
13	462	3.13	3.964	0
14	492	3.13	3.587	1
15	522	3.13	3.438	2
16	552	3.13	3.694	3
17	582	3.13	3.526	4
18	612	2.992	3.43	0
19	642	2.992	3.217	1
20	672	2.992	3.246	2
21	702	2.992	3.185	3
22	732	2.992	3.204	4
23	762	2.992	3.223	5
24	792	2.992	3.388	6
25	822	2.992	3.101	7
26	852	2.992	3.03	8

27	882	2.992	3.056	9
28	912	2.992	3.028	10
29	942	2.992	3.154	11
30	972	2.992	3.162	12

t2 =

3.8768

Elapsed time is 0.103391 seconds.

Generation	Best f-count	Best f(x)	Mean f(x)	Stall Generations
1	60	5.782	6.346	0
2	90	5.652	6.224	0
3	120	4.274	6.035	0
4	150	3.725	5.959	0
5	180	3.725	5.323	1
6	210	3.725	4.888	0
7	240	3.35	4.509	0
8	270	3.35	4.449	1
9	300	3.35	4.242	2
10	330	3.35	4.285	0
11	360	3.283	4.109	0
12	390	3.283	4.275	1
13	420	3.283	3.794	2
14	450	3.283	3.912	0
15	480	3.246	3.635	0
16	510	3.246	3.518	0
17	540	3.246	3.529	1
18	570	3.246	3.61	2
19	600	3.246	3.699	3
20	630	3.246	3.586	4

21	660	3.246	3.459	5
22	690	3.246	3.377	6
23	720	3.246	3.369	7
24	750	3.246	3.412	8
25	780	3.246	3.506	9
26	810	3.246	3.442	10
27	840	3.246	3.929	11
28	870	3.246	3.856	12
29	900	3.246	3.753	13
30	930	3.246	3.507	14

t3 =

3.8040

Elapsed time is 3.808372 seconds.

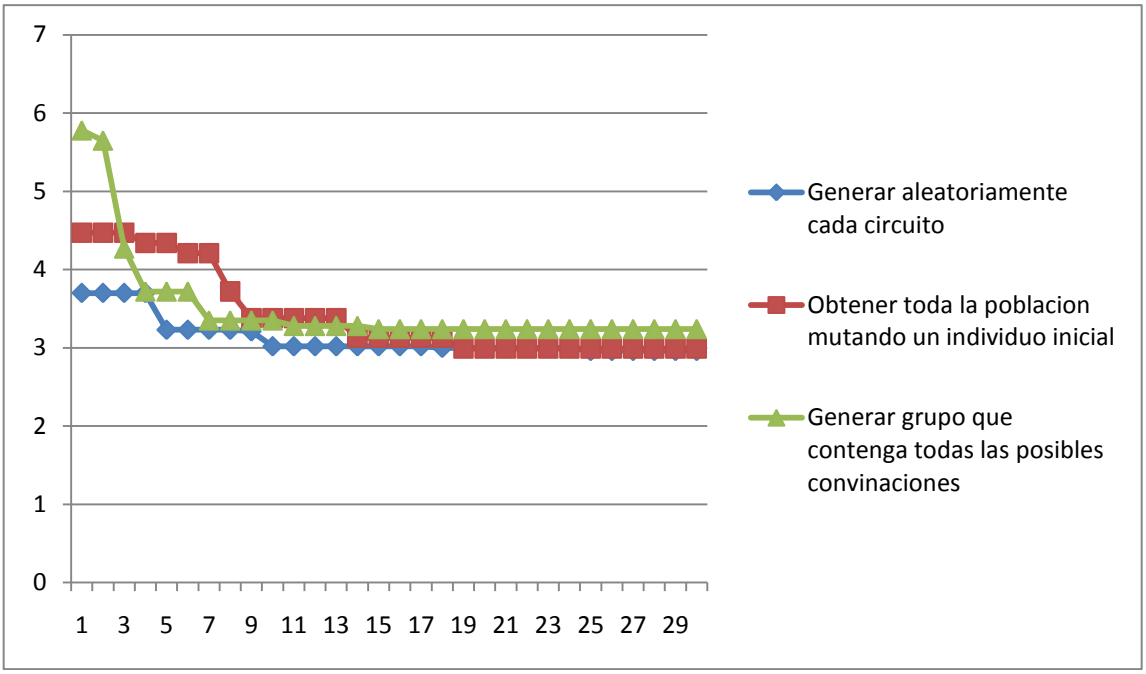


Figura 3.21: Comparación entre métodos de creación de población inicial para primeras 30 generaciones (circuito de 30 barras).

Ejemplo 2:

ga_mpc =

version: '2'

```
    baseMVA: 100
        bus: [30x13 double]
        gen: [6x21 double]
     branch: [41x13 double]
      areas: [3x2 double]
    gencost: [6x7 double]
Elapsed time is 1.609655 seconds.
```

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	60	4.384	7.311	0
2	90	3.766	6.468	0
3	120	3.766	3.875e+004	1
4	150	3.766	5.069	2
5	180	3.766	5.2	3
6	210	3.746	4.738	0
7	240	3.746	4.571	1
8	270	3.469	4.562	0
9	300	2.975	337.8	0
10	330	2.975	3.692	1
11	360	2.967	3.709	0
12	390	2.967	3.622	1
13	420	2.967	3.38	2
14	450	2.967	3.25	3
15	480	2.967	3.128	4
16	510	2.967	3.056	5
17	540	2.967	3.091	6
18	570	2.967	3.12	7
19	600	2.967	3.099	8
20	630	2.967	3.106	9

21	660	2.967	3.095	10
22	690	2.967	3.11	11
23	720	2.967	3.918	12
24	750	2.967	2720	13
25	780	2.967	3136	14
26	810	2.967	435.6	0
27	840	2.967	3.144	1
28	870	2.967	3.245	2
29	900	2.967	3.025	3
30	930	2.967	3.016	4

t1 =

5.2658

Elapsed time is 0.072064 seconds.

		Best	Mean	Stall
Generation	f-count	f(x)	f(x)	Generations
1	90	6.203	341.3	0
2	120	6.203	340.8	1
3	150	5.225	341	0
4	180	5.225	6.991	1
5	210	5.045	6.027	0
6	240	5.045	6.013	1
7	270	5.045	5.641	2
8	300	5.045	5.33	3
9	330	4.656	5.137	0
10	360	4.656	5.086	1
11	390	4.626	4.99	0
12	420	4.238	338.4	0
13	450	4.111	338.3	0
14	480	4.111	338.2	1

15	510	3.724	4.875	0
16	540	3.617	4.469	0
17	570	3.617	4.19	1
18	600	3.617	3.932	2
19	630	3.617	3.848	3
20	660	3.606	3.731	0
21	690	3.562	3.689	0
22	720	3.562	337.3	1
23	750	3.562	3.808	2
24	780	3.562	3.827	3
25	810	3.556	3.916	0
26	840	3.521	4.004	0
27	870	3.507	3.85	0
28	900	3.507	3.778	1
29	930	3.507	3.559	2
30	960	3.507	3.573	3

t2 =

4.3163

Elapsed time is 0.101728 seconds.

		Best	Mean	Stall
Generation	f-count	f(x)	f(x)	Generations
1	60	4.907	6.978	0
2	90	4.907	6.653	1
3	120	4.489	6.143	0
4	150	4.489	6.012	1
5	180	4.367	5.72	0
6	210	4.367	5.423	1
7	240	4.14	338.7	0
8	270	3.643	339.1	0

9	300	3.643	4.911	1
10	330	3.405	4.385	0
11	360	3.405	4.266	1
12	390	3.405	4.24	2
13	420	3.405	4.283	3
14	450	3.405	3.644	0
15	480	3.405	3.837	1
16	510	3.381	3.548	0
17	540	3.381	3.51	1
18	570	3.381	3.6	2
19	600	3.381	3.609	3
20	630	3.381	3.681	4
21	660	3.381	3.414	5
22	690	3.381	3.458	6
23	720	3.381	3.482	7
24	750	3.381	3.476	8
25	780	3.119	3.39	0
26	810	3.119	3.4	1
27	840	3.119	3.332	2
28	870	3.119	3.28	3
29	900	3.119	3.239	4
30	930	3.119	3.321	5

t3 =

4.6002

Elapsed time is 4.604113 seconds.

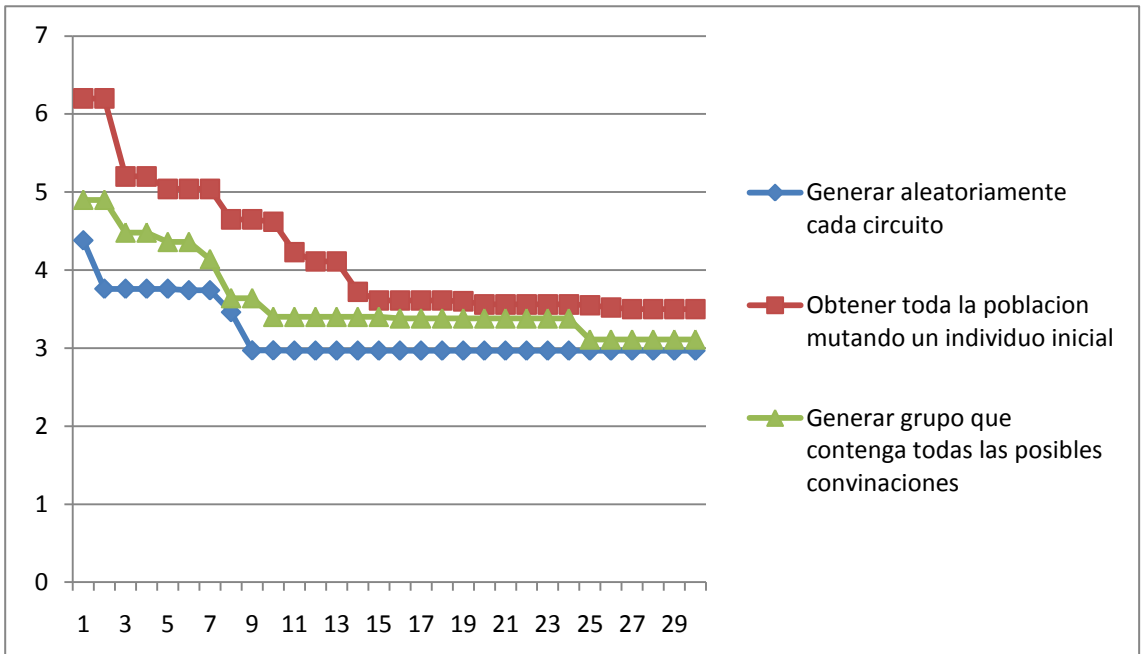


Figura 3.22: Comparación entre métodos de creación de población inicial para primeras 30 poblaciones (circuito de 30 barras).

Ejemplo 3:

ga_mpc =

version: '2'
baseMVA: 100
bus: [30x13 double]
gen: [6x21 double]
branch: [41x13 double]
areas: [3x2 double]
gencost: [6x7 double]

Elapsed time is 1.592499 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	60	4.183	1059	0

2	90	4.183	1398	1
3	120	4.183	346.7	2
4	150	3.793	4.603	0
5	180	3.793	4.314	1
6	210	3.793	4.192	2
7	240	3.793	4.049	3
8	270	3.512	3.947	0
9	300	3.512	4.076	1
10	330	3.512	3.803	2
11	360	3.512	3.763	3
12	390	3.512	3.702	4
13	420	3.512	3.598	5
14	450	3.512	3.569	6
15	480	3.512	3.658	7
16	510	3.512	3.662	8
17	540	3.512	3.766	9
18	570	3.512	3.759	10
19	600	3.512	3.549	11
20	630	3.512	3.519	12
21	660	3.512	3.526	13
22	690	3.512	3.607	14
23	720	3.512	3.579	15
24	750	3.512	3.543	16
25	780	3.512	3.563	17
26	810	3.512	3.513	18
27	840	3.512	3.636	19
28	870	3.428	3.566	0
29	900	3.428	4.274	1
30	930	3.428	4.342	2

t1 =

6.0193

Elapsed time is 0.075438 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	96	4.11	5.68	0
2	126	3.929	7.287	0
3	156	3.929	5.38	1
4	186	3.663	4.976	0
5	216	3.663	4.536	1
6	246	3.663	4.303	2
7	276	3.663	4.027	3
8	306	3.47	4.264	0
9	336	3.47	4.158	1
10	366	3.47	4.065	2
11	396	3.47	3.975	3
12	426	3.343	3.811	0
13	456	3.343	3.679	1
14	486	3.09	4.155	0
15	516	3.09	3.843	1
16	546	3.09	3.729	2
17	576	3.034	3.491	0
18	606	3.034	3.631	1
19	636	3.034	3.233	2
20	666	3.034	3.376	3
21	696	3.034	3.774	4
22	726	3.034	3.375	5
23	756	3.034	3.187	6
24	786	3.034	3.284	7

25	816	3.034	3.161	8
26	846	3.034	3.189	9
27	876	3.034	3.053	10
28	906	3.034	3.036	11
29	936	3.034	3.036	12
30	966	2.96	3.201	0

t2 =

5.0226

Elapsed time is 0.102304 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	60	4.711	796.6	0
2	90	4.711	5.531	1
3	120	4.569	420.8	0
4	150	4.456	420.4	0
5	180	4.456	5.073	1
6	210	4.456	5.035	2
7	240	4.456	4.649	3
8	270	4.456	4.578	4
9	300	4.456	4.526	5
10	330	4.456	4.552	6
11	360	4.456	4.536	7
12	390	4.357	4.575	0
13	420	4.29	4.555	0
14	450	4.29	4.627	1
15	480	4.29	4.537	2
16	510	4.29	4.38	3
17	540	4.29	4.367	4
18	570	4.29	4.785	5

19	600	4.29	4.633	6
20	630	4.29	4.64	7
21	660	4.29	4.644	8
22	690	4.121	1.961e+004	0
23	720	4.121	493.3	1
24	750	4.121	493.3	2
25	780	4.075	493.3	0
26	810	4.056	493.4	0
27	840	4.056	4.314	1
28	870	4.056	4.29	2
29	900	4.056	4.226	3
30	930	3.516	4.303	0

t3 =

7.2979

Elapsed time is 7.302234 seconds.

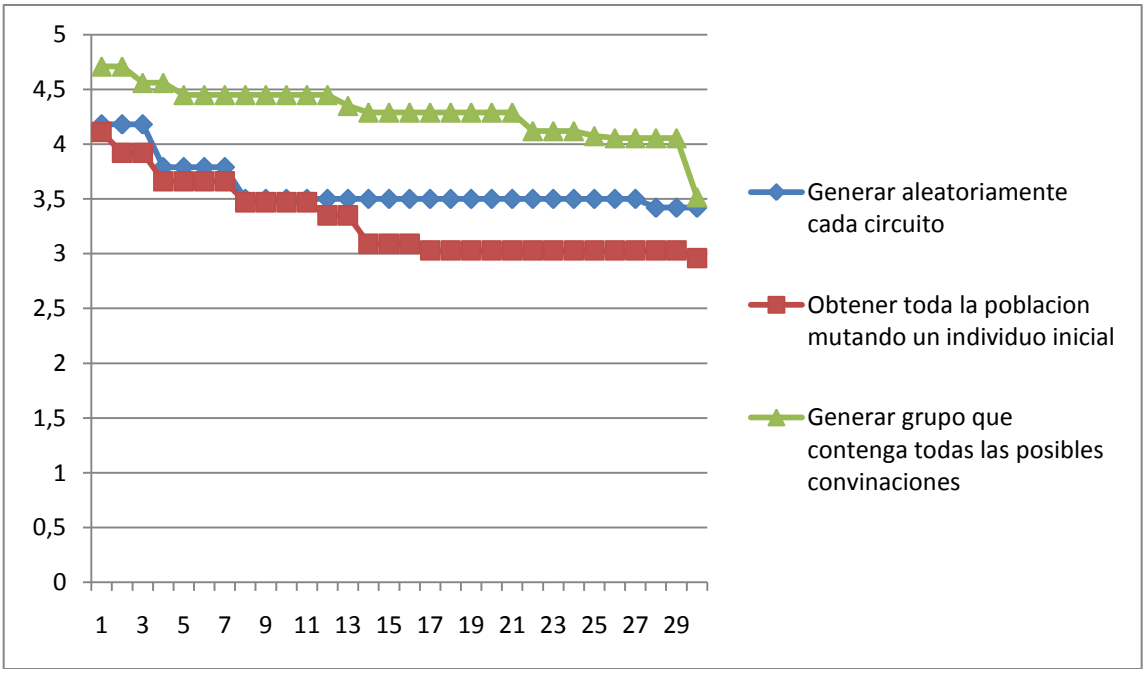


Figura 3.23: Comparación entre métodos de creación de población inicial para primeras 30 generaciones (circuito de 30 barras).

Ejemplo	1	2	3
Método usado			
Generar aleatoriamente cada circuito	1.59seg	1.609seg	1.592seg
Obtener toda la población mutando un individuo	0.07seg	0.072seg	0.075seg
Generar grupo que contenga todas las posibles combinaciones	0.1seg	0.101seg	0.102seg

Tabla 1: Comparación de métodos de creación de población

Estas tablas se confeccionaron una por cada tipo de circuito y dan una muestra de la diferencia de tiempo con que se crea la población inicial entre los tres métodos estudiados y que tanto el método de obtener toda la población mutando un individuo como el método de generar un grupo que contenga todas las posibles combinaciones son mucho más rápidas que el método de generar aleatoriamente cada circuito.

1.2.2 Para circuitos de 57 barras:

Ejemplo 1:

ga_mpc =

version: '2'

baseMVA: 100

bus: [57x13 double]

gen: [7x21 double]

branch: [80x13 double]

gencost: [7x7 double]

Elapsed time is 9.355385 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	114	1.006e+004	5.629e+004	0
2	171	1.006e+004	3.782e+004	0
3	228	1.006e+004	3.73e+004	1
4	285	1.006e+004	6.186e+004	2
5	342	1.005e+004	3.237e+005	0
6	399	1.005e+004	3.324e+005	1
7	456	54.2	2.829e+004	0
8	513	54.2	7.863e+007	1
9	570	54.2	4.423e+004	2
10	627	54.2	5.455e+004	3
11	684	53.62	5.842e+006	0
12	741	53.37	5.075e+004	0
13	798	53.37	8.203e+004	1
14	855	49.46	1.086e+005	0
15	912	49.46	4.942e+004	1
16	969	49.46	5.713e+005	2
17	1026	48.87	5.532e+005	0
18	1083	47.98	5056	0
19	1140	47.98	6.949e+005	1
20	1197	47.98	2.802e+005	2

21	1254	47.39	1639	0
22	1311	47.39	2002	1
23	1368	47.39	3178	2
24	1425	47.39	1.161e+005	3
25	1482	47.39	4.986e+004	4
26	1539	46.95	2084	0
27	1596	44.26	1.706e+004	0
28	1653	44.26	1.443e+004	1
29	1710	43.91	1.487e+004	0
30	1767	43.91	1.093e+005	1

t1 =

31.5123

Elapsed time is 0.220471 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	222	1.293e+004	6.411e+004	0
2	279	1.278e+004	1.161e+007	0
3	336	1.278e+004	2.803e+004	1
4	393	1.278e+004	2.762e+005	2
5	450	1.266e+004	2.68e+005	0
6	507	1.257e+004	3.967e+006	0
7	564	1.24e+004	7.042e+005	0
8	621	1.151e+004	4.244e+005	0

9	678	1.151e+004	4.481e+005	0
10	735	1.151e+004	4.856e+005	1
11	792	1.008e+004	2.717e+005	0
12	849	1.008e+004	6.407e+004	0
13	906	1.006e+004	6.726e+004	0
14	963	1.006e+004	4.955e+005	1
15	1020	1.006e+004	5.11e+005	0
16	1077	1.006e+004	4.989e+005	1
17	1134	1.006e+004	6.838e+005	2
18	1191	1.006e+004	6.415e+004	3
19	1248	1.006e+004	3.449e+004	4
20	1305	1.006e+004	5.791e+004	5
21	1362	1.006e+004	8.322e+004	6
22	1419	1.006e+004	6.536e+004	7
23	1476	1.006e+004	6.696e+005	8
24	1533	1.006e+004	6.712e+005	9
25	1590	1.006e+004	2.414e+005	10
26	1647	1.006e+004	3.831e+007	11
27	1704	1.006e+004	7.665e+004	12
28	1761	1.006e+004	2.915e+004	0
29	1818	1.006e+004	1.813e+004	1
30	1875	1.006e+004	2.523e+004	0

t2 =

26.0176

Elapsed time is 0.448388 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	114	1.128e+004	3.355e+005	0
2	171	1.008e+004	1.868e+005	0
3	228	1.008e+004	5.812e+005	1
4	285	1.008e+004	3.194e+005	0
5	342	1.008e+004	3.707e+006	1
6	399	1.007e+004	3.703e+006	0
7	456	1.007e+004	3.792e+006	1
8	513	1.007e+004	1.477e+005	2
9	570	1.007e+004	8.032e+004	3
10	627	1.007e+004	4.04e+004	0
11	684	1.007e+004	2.359e+004	0
12	741	1.005e+004	2.602e+004	0
13	798	1.005e+004	1.28e+004	1
14	855	53.53	1.849e+004	0
15	912	52.02	2.734e+004	0
16	969	46.65	2.107e+004	0
17	1026	46.29	4.784e+004	0
18	1083	46.29	1.048e+007	1
19	1140	44.82	1.052e+007	0
20	1197	44.82	7193	1
21	1254	44.82	4298	2

22	1311	44.82	1.283e+004	3
23	1368	44.82	1.368e+004	4
24	1425	44.07	3861	0
25	1482	44.07	1.938e+004	1
26	1539	44.07	2.944e+004	2
27	1596	44.07	1.107e+004	3
28	1653	44.07	1.128e+004	4
29	1710	44.07	2.914e+004	5
30	1767	44.07	4.415e+006	6

t3 =

23.6621

Elapsed time is 23.667739 seconds.

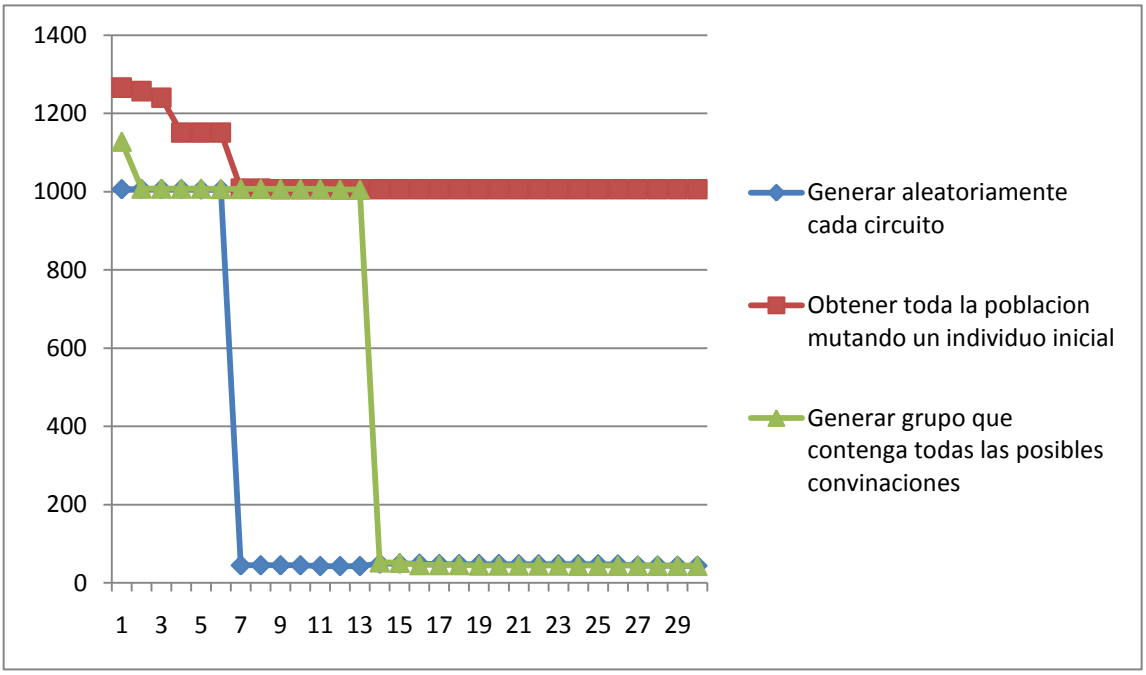


Figura 3.24: Comparación entre métodos de creación de población inicial para primeras 30 generaciones (circuito de 57 barras).

Ejemplo 2:

ga_mpc =

version: '2'

baseMVA: 100

bus: [57x13 double]

gen: [7x21 double]

branch: [80x13 double]

gencost: [7x7 double]

Elapsed time is 9.164145 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	114	91.38	5.258e+004	0
2	171	86.99	5.242e+004	0
3	228	86.99	3.481e+004	1
4	285	86.99	3.381e+004	2
5	342	85.4	1.293e+004	0
6	399	84.03	8.573e+004	0
7	456	84.03	1.398e+005	1
8	513	74.03	1.37e+004	0
9	570	72.12	1.435e+004	0
10	627	72.12	2.11e+004	0
11	684	72.07	1.047e+004	0
12	741	72.07	7239	1
13	798	70.74	1.027e+005	0

14	855	69.69	6.243e+004	0
15	912	69.69	3021	1
16	969	69.14	1823	0
17	1026	67.67	3862	0
18	1083	67.67	6.01e+004	1
19	1140	67.67	3.947e+004	2
20	1197	67.67	1.179e+004	3
21	1254	67.67	1.342e+004	4
22	1311	67.67	9571	5
23	1368	67.66	8971	0
24	1425	61.63	8.614e+005	0
25	1482	61.63	1.406e+006	1
26	1539	61.63	1.414e+006	2
27	1596	61.63	5.652e+005	0
28	1653	54.75	6107	0
29	1710	54.75	2.777e+004	1
30	1767	54.74	3.524e+004	0

t1 =

34.6577

Elapsed time is 0.220810 seconds.

		Best	Mean	Stall
Generation	f-count	f(x)	f(x)	Generations
1	219	1.007e+004	4.473e+004	0
2	276	1.007e+004	5.445e+004	1

3	333	1.007e+004	3.235e+004	2
4	390	1.007e+004	1.464e+005	0
5	447	1.007e+004	2.113e+004	1
6	504	1.007e+004	3.487e+004	0
7	561	1.007e+004	1.303e+004	0
8	618	1.006e+004	1.361e+004	0
9	675	1.006e+004	3.08e+004	0
10	732	1.006e+004	3.592e+004	0
11	789	1.005e+004	3.059e+004	0
12	846	1.005e+004	2.663e+004	1
13	903	1.005e+004	1.952e+004	0
14	960	1.005e+004	2.156e+004	1
15	1017	1.005e+004	1.459e+004	2
16	1074	1.005e+004	1.82e+004	0
17	1131	1.005e+004	1.917e+004	0
18	1188	1.005e+004	1.745e+004	0
19	1245	1.005e+004	1.416e+004	1
20	1302	1.005e+004	1.797e+004	0
21	1359	1.005e+004	1.582e+004	1
22	1416	1.005e+004	1.041e+004	2
23	1473	1.005e+004	1.088e+004	3
24	1530	50.78	1.287e+004	0
25	1587	50.78	9655	1
26	1644	50.78	2.548e+004	2

27	1701	47.06	2.198e+004	0
28	1758	47.06	4235	1
29	1815	47.06	5084	2
30	1872	47.06	1.515e+008	3

t2 =

22.5230

Elapsed time is 0.433738 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	114	1.018e+004	2.302e+005	0
2	171	1.015e+004	1.8e+007	0
3	228	1.007e+004	1.803e+007	0
4	285	1.007e+004	1.379e+005	0
5	342	1.007e+004	2.102e+005	0
6	399	1.007e+004	1.875e+012	1
7	456	1.007e+004	1.875e+012	2
8	513	1.007e+004	6.95e+004	3
9	570	1.006e+004	2.13e+004	0
10	627	68.65	1.463e+004	0
11	684	68.36	1.137e+004	0
12	741	68.36	1.344e+004	1
13	798	67.73	1.393e+004	0
14	855	67.73	1.216e+004	1
15	912	67.73	9985	2

16	969	59.61	1.626e+004	0
17	1026	57.9	5376	0
18	1083	57.9	6227	1
19	1140	57.9	1.25e+005	2
20	1197	57.9	1.264e+005	3
21	1254	52.91	4258	0
22	1311	52.39	1.061e+005	0
23	1368	49.08	1.082e+005	0
24	1425	47.98	3.612e+004	0
25	1482	47.89	5.874e+004	0
26	1539	47.89	5.892e+004	1
27	1596	47.32	1.015e+007	0
28	1653	47.32	1.011e+007	1
29	1710	47.32	3453	2
30	1767	47.32	3425	3

t3 =

20.3889

Elapsed time is 20.394837 seconds.

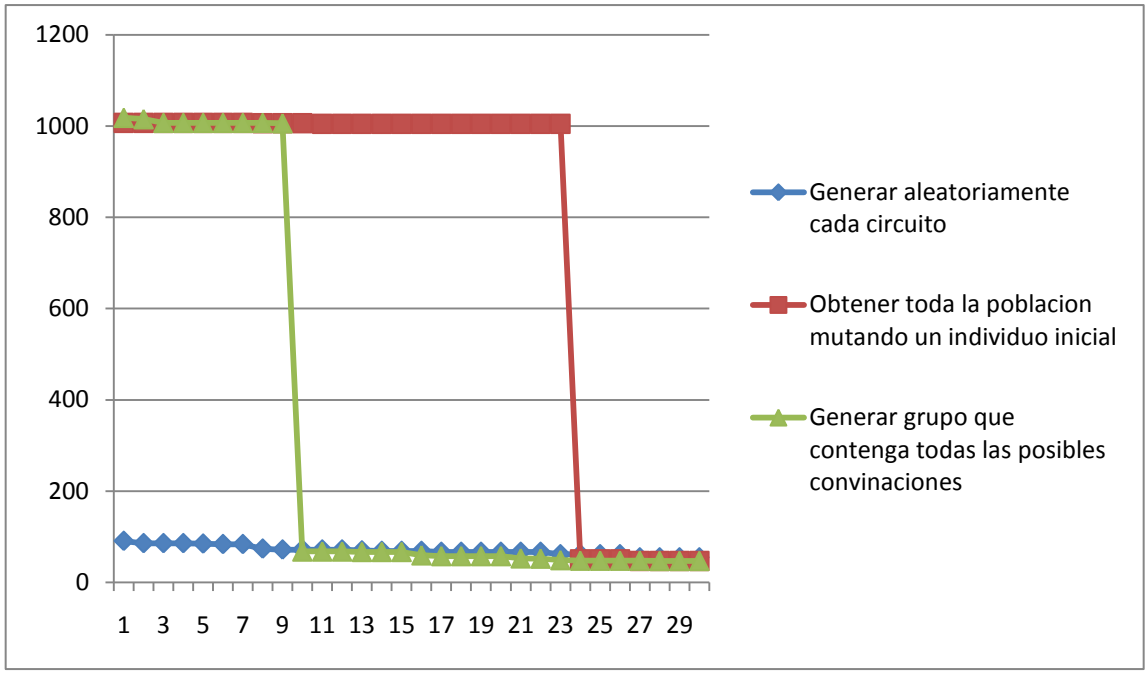


Figura 3.25: Comparación entre métodos de creación de población inicial para primeras 30 generaciones (circuito de 57 barras).

Ejemplo 3:

ga_mpc =

version: '2'

baseMVA: 100

bus: [57x13 double]

gen: [7x21 double]

branch: [80x13 double]

gencost: [7x7 double]

Elapsed time is 9.173334 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	114	69.6	2.884e+006	0

2	171	69.6	9.097e+005	1
3	228	69.6	7.271e+005	2
4	285	69.23	1.798e+005	0
5	342	61.35	7.023e+005	0
6	399	61.31	7.039e+005	0
7	456	57.96	6.412e+005	0
8	513	57.96	6.203e+005	1
9	570	57.87	6.182e+005	0
10	627	47.53	3.455e+005	0
11	684	46.66	9.869e+004	0
12	741	44.57	1.434e+005	0
13	798	44.57	6.138e+004	1
14	855	44.57	7586	2
15	912	44.57	1972	3
16	969	41.91	935.7	0
17	1026	41.91	1264	1
18	1083	41.62	1065	0
19	1140	41.62	4223	1
20	1197	41.62	3776	2
21	1254	41.62	1.028e+005	3
22	1311	41.62	9.933e+004	4
23	1368	41.62	5.56e+004	5
24	1425	41.62	3.558e+004	6
25	1482	41.62	5.402e+004	7

26	1539	41.48	2.862e+005	0
27	1596	41.48	2.928e+005	1
28	1653	41.48	1.093e+004	2
29	1710	41.48	8.06e+004	3
30	1767	41.07	7.318e+004	0

t1 =

28.2307

Elapsed time is 0.218034 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	228	1.153e+004	2.624e+006	0
2	285	1.153e+004	7.462e+005	1
3	342	1.011e+004	5.178e+005	0
4	399	1.011e+004	6.231e+005	1
5	456	1.01e+004	2.856e+005	0
6	513	1.01e+004	5.565e+011	1
7	570	1.01e+004	4.22e+004	0
8	627	1.01e+004	1.662e+004	1
9	684	1.009e+004	1.332e+005	0
10	741	1.009e+004	3.319e+004	1
11	798	1.009e+004	1.472e+007	0
12	855	1.009e+004	3.463e+005	0
13	912	1.009e+004	3.518e+004	1
14	969	1.009e+004	3.745e+004	2

15	1026	1.009e+004	1.421e+005	3
16	1083	1.008e+004	1.377e+005	0
17	1140	1.007e+004	1.268e+004	0
18	1197	1.007e+004	6.124e+004	1
19	1254	1.007e+004	6.966e+004	2
20	1311	1.007e+004	1.699e+004	3
21	1368	1.006e+004	3.176e+004	0
22	1425	1.006e+004	2.449e+004	1
23	1482	1.006e+004	2.24e+004	2
24	1539	1.006e+004	2.07e+004	0
25	1596	1.006e+004	2.08e+004	1
26	1653	1.006e+004	1.482e+004	2
27	1710	1.005e+004	2.339e+007	0
28	1767	1.005e+004	1.323e+004	1
29	1824	1.005e+004	1.248e+004	2
30	1881	1.005e+004	8.907e+004	3

t2 =

40.0026

Elapsed time is 0.427722 seconds.

		Best	Mean	Stall
Generation	f-count	f(x)	f(x)	Generations
1	114	1.151e+004	2.873e+009	0
2	171	1.147e+004	1.458e+005	0
3	228	1.147e+004	5.547e+004	1

4	285	1.017e+004	3.097e+005	0
5	342	1.017e+004	7.277e+005	1
6	399	1.008e+004	1.746e+004	0
7	456	1.008e+004	3.222e+004	1
8	513	1.006e+004	3.822e+004	0
9	570	1.006e+004	1.589e+004	1
10	627	1.006e+004	1.71e+004	2
11	684	1.006e+004	1.774e+004	3
12	741	1.006e+004	1.381e+004	4
13	798	1.005e+004	8.407e+005	0
14	855	1.005e+004	1.199e+004	1
15	912	1.005e+004	3.916e+010	0
16	969	1.005e+004	3.916e+010	1
17	1026	47.93	3.916e+010	0
18	1083	47.93	4.871e+004	1
19	1140	47.93	2.572e+004	2
20	1197	47.93	6.295e+009	3
21	1254	47.93	6.297e+009	4
22	1311	47.93	6.295e+009	5
23	1368	47.93	1.203e+004	6
24	1425	47.93	6.295e+009	7
25	1482	47.81	6.295e+009	0
26	1539	45.48	6.297e+009	0
27	1596	45.48	6.295e+009	1

28	1653	45.35	5809	0
29	1710	45.35	1.54e+004	1
30	1767	43.35	2422	0

t3 =

24.7116

Elapsed time is 24.715768 seconds.

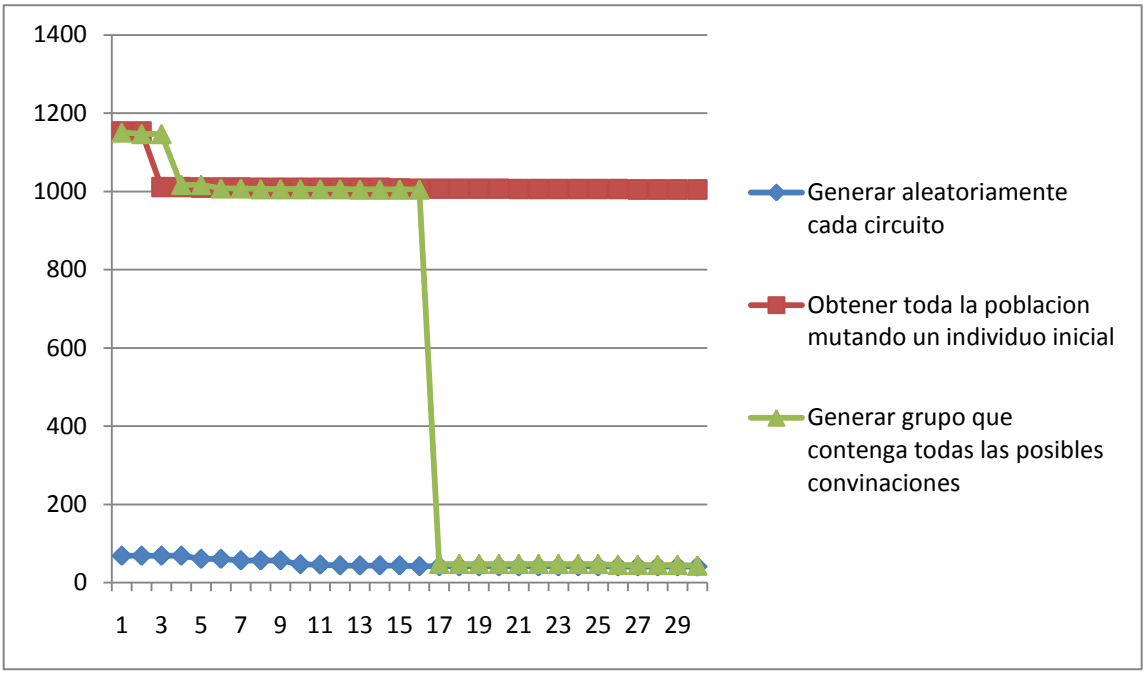


Figura 3.26: Comparación entre métodos de creación de población inicial para primeras 30 generaciones (circuito de 57 barras).

Ejemplo	1	2	3
Método usado			
Generar aleatoriamente cada circuito	9.355seg	9.164seg	9.173seg
Obtener toda la población	0.22seg	0.22seg	0.21seg

mutando un individuo			
Generar grupo que contenga todas las posibles combinaciones	0.44seg	0.433seg	0.427seg

Tabla 2: Comparación de métodos de creación de población.

1.2.3 Para circuito de 118 barras:

Ejemplo 1:

ga_mpc =

version: '2'

baseMVA: 100

bus: [118x13 double]

gen: [54x21 double]

branch: [186x13 double]

gencost: [54x7 double]

Elapsed time is 84.953116 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	236	1.036e+004	4.238e+005	0
2	354	689	1.634e+005	0
3	472	534.7	2.411e+004	0
4	590	522.5	1.961e+006	0
5	708	469.5	1.503e+005	0

6	826	469.5	5.228e+004	1
7	944	436.9	6.322e+004	0
8	1062	420.7	1.078e+005	0
9	1180	398	4.994e+004	0
10	1298	380.7	3.479e+005	0
11	1416	358.9	7613	0
12	1534	358.9	1.667e+004	1
13	1652	349.1	6186	0
14	1770	331.6	1.457e+004	0
15	1888	331.6	2.264e+004	1
16	2006	306.4	2.446e+004	0
17	2124	300.2	1.11e+004	0
18	2242	298.7	5255	0
19	2360	287	5462	0
20	2478	284.6	5001	0
21	2596	264.6	4780	0
22	2714	264.6	1.267e+004	1
23	2832	257.7	9.73e+006	0
24	2950	257.7	3254	1
25	3068	245.6	2806	0
26	3186	240.5	2801	0
27	3304	233.8	9405	0
28	3422	233.2	4561	0
29	3540	229.2	4410	0

30	3658	214.5	1.14e+006	0

t1 =

214.1580

Elapsed time is 1.032167 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	499	1.081e+004	1.383e+010	0
2	617	1.081e+004	1.383e+010	1
3	735	1.081e+004	4.922e+004	2
4	853	1.08e+004	1.264e+005	0
5	971	1.069e+004	1.016e+005	0
6	1089	1.069e+004	2.381e+004	1
7	1207	1.054e+004	4.815e+005	0
8	1325	1.054e+004	5.532e+004	0
9	1443	1.054e+004	7.07e+004	1
10	1561	1.054e+004	1.611e+005	2
11	1679	1.052e+004	5.281e+005	0
12	1797	1.051e+004	5.441e+005	0
13	1915	1.051e+004	5.04e+005	1
14	2033	1.051e+004	8.726e+005	0
15	2151	1.051e+004	1.696e+004	0
16	2269	1.051e+004	1.557e+004	0
17	2387	1.051e+004	2.428e+004	0
18	2505	1.051e+004	4.877e+004	1

19	2623	1.051e+004	4.982e+005	0
20	2741	1.048e+004	1.041e+005	0
21	2859	1.048e+004	5.863e+004	1
22	2977	1.047e+004	2.224e+004	0
23	3095	1.047e+004	1.922e+004	0
24	3213	1.047e+004	8.158e+007	0
25	3331	1.047e+004	2.603e+004	0
26	3449	1.047e+004	5.325e+004	0
27	3567	1.043e+004	2.817e+005	0
28	3685	1.042e+004	3.535e+004	0
29	3803	1.04e+004	1.828e+004	0
30	3921	416.6	5.998e+004	0

t2 =

144.0254

Elapsed time is 4.234924 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	236	488.3	1.646e+007	0
2	354	485.8	1.202e+006	0
3	472	395.6	6.236e+004	0
4	590	385.1	1.709e+005	0
5	708	367.9	3.294e+004	0
6	826	354.1	3.703e+004	0

7	944	320.1	1.357e+005	0
8	1062	318.7	1.257e+004	0
9	1180	318.7	1.095e+004	1
10	1298	301.6	1.404e+004	0
11	1416	294.7	1.46e+005	0
12	1534	288.4	1.473e+004	0
13	1652	283.3	3.782e+004	0
14	1770	275.2	1.571e+004	0
15	1888	263.2	3.707e+004	0
16	2006	259	7.405e+004	0
17	2124	238.7	2092	0
18	2242	238	3.006e+005	0
19	2360	227	3023	0
20	2478	227	1.872e+004	0
21	2596	214.9	4.038e+004	0
22	2714	214.9	6557	1
23	2832	202.4	1.124e+004	0
24	2950	202	1.848e+004	0
25	3068	201.8	2.422e+004	0
26	3186	199.3	2.002e+004	0
27	3304	199.2	1.978e+004	0
28	3422	193.1	5.496e+004	0
29	3540	193.1	2.501e+004	1
30	3658	192.8	2.674e+004	0

t3 =

131.3779

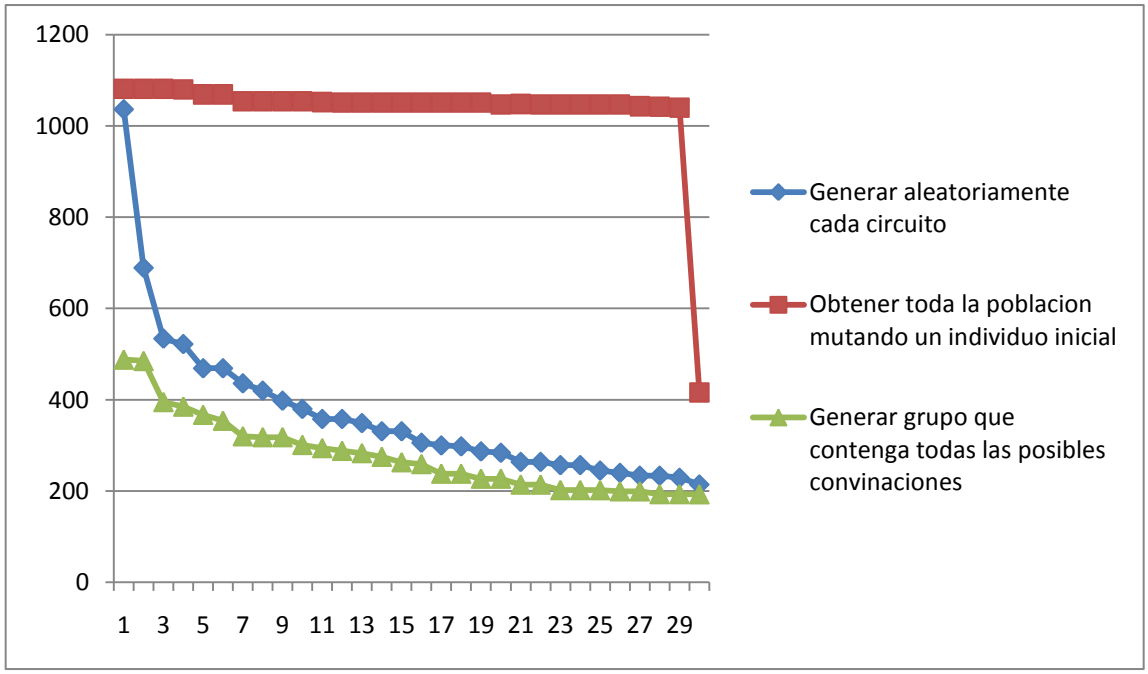


Figura 3.27: Comparación entre métodos de creación de población inicial para primeras 30 generaciones (circuito de 118 barras).

Ejemplo 2

ga_mpc =

version: '2'

baseMVA: 100

bus: [118x13 double]

gen: [54x21 double]

branch: [186x13 double]

gencost: [54x7 double]

Elapsed time is 85.650955 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	236	1.057e+004	4.899e+005	0
2	354	680.9	6.36e+006	0
3	472	654.8	7.642e+005	0
4	590	654.8	7.239e+007	1
5	708	574.8	7.238e+007	0
6	826	548.8	9.889e+005	0
7	944	533.6	1.142e+004	0
8	1062	523.2	2.548e+005	0
9	1180	509.1	3.992e+005	0
10	1298	481.5	5.759e+004	0
11	1416	384.8	7053	0
12	1534	384.8	8471	1
13	1652	352.2	8957	0
14	1770	321.9	2.749e+006	0
15	1888	310.5	9662	0
16	2006	283.2	2.012e+004	0
17	2124	283.2	3.34e+004	1
18	2242	278.8	5.154e+004	0
19	2360	260.7	2.134e+004	0
20	2478	257.3	2e+004	0

21	2596	256.8	1.188e+005	0
22	2714	242.7	4.253e+005	0
23	2832	237.7	1.032e+004	0
24	2950	235.9	7.789e+004	0
25	3068	232.7	2.47e+004	0
26	3186	232.7	3.646e+005	1
27	3304	228.2	3.888e+005	0
28	3422	228.2	1.747e+009	1
29	3540	220	3792	0
30	3658	219	1.159e+004	0

t1 =

215.4544

Elapsed time is 1.050463 seconds.

		Best	Mean	Stall
Generation	f-count	f(x)	f(x)	Generations
1	520	1.062e+004	5.793e+005	0
2	638	1.059e+004	4.025e+005	0
3	756	1.056e+004	3.111e+004	0
4	874	1.055e+004	5.467e+004	0
5	992	1.05e+004	4.601e+004	0
6	1110	1.05e+004	6.117e+004	1
7	1228	1.049e+004	1.916e+005	0
8	1346	1.048e+004	1.471e+005	0
9	1464	1.046e+004	1.732e+004	0

10	1582	1.043e+004	5.007e+004	0
11	1700	1.041e+004	3.563e+008	0
12	1818	1.04e+004	3.563e+008	0
13	1936	1.039e+004	7.509e+007	0
14	2054	1.039e+004	2.768e+004	0
15	2172	1.037e+004	3.209e+004	0
16	2290	1.036e+004	1.922e+004	0
17	2408	1.036e+004	2.588e+004	1
18	2526	1.036e+004	2.725e+004	0
19	2644	1.035e+004	2.266e+004	0
20	2762	1.035e+004	2.17e+004	1
21	2880	1.034e+004	7.51e+009	0
22	2998	1.033e+004	7.511e+009	0
23	3116	1.033e+004	7.51e+009	1
24	3234	1.033e+004	5.674e+004	0
25	3352	1.033e+004	5.917e+004	0
26	3470	1.03e+004	5.205e+004	0
27	3588	1.03e+004	3.032e+004	1
28	3706	1.03e+004	1.504e+004	2
29	3824	1.03e+004	1.122e+005	3
30	3942	450.1	1.278e+005	0

t2 =

149.7960

Elapsed time is 4.336158 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	236	447.8	3.089e+004	0
2	354	431	2.232e+005	0
3	472	425.7	1.243e+006	0
4	590	399.1	3.964e+004	0
5	708	394	7283	0
6	826	372.4	6995	0
7	944	362.5	4440	0
8	1062	327.5	2.144e+006	0
9	1180	327.5	1.723e+005	1
10	1298	325.2	1.702e+005	0
11	1416	312.6	4619	0
12	1534	301.3	2.134e+006	0
13	1652	298.1	2.135e+006	0
14	1770	298.1	6869	1
15	1888	295.4	3280	0
16	2006	286.9	1.356e+004	0
17	2124	282.9	2917	0
18	2242	280.6	2473	0
19	2360	279.7	2213	0
20	2478	274.1	2554	0
21	2596	266	1970	0
22	2714	266	1.556e+004	1

23	2832	257.5	1.657e+004	0
24	2950	256.6	3424	0
25	3068	247.8	3335	0
26	3186	247.5	3329	0
27	3304	247	2512	0
28	3422	239.4	4560	0
29	3540	222.9	3772	0
30	3658	222.9	3665	1

t3 =

131.9600

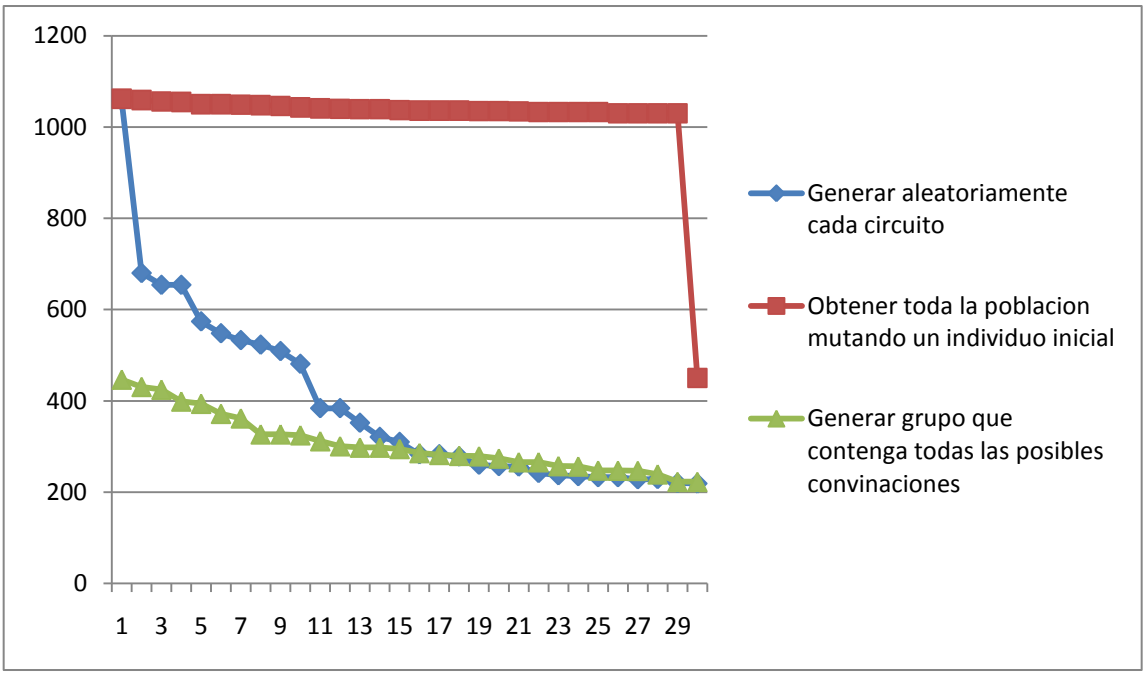


Figura 3.28: Comparación entre métodos de creación de población inicial para primeras 30 generaciones (circuito de 118 barras).

Ejemplo 3

ga_mpc =

version: '2'

baseMVA: 100

bus: [118x13 double]

gen: [54x21 double]

branch: [186x13 double]

gencost: [54x7 double]

Elapsed time is 86.000963 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	236	705.9	1.75e+006	0
2	354	604	1.767e+006	0
3	472	586.2	3.979e+004	0
4	590	473.8	2.316e+004	0
5	708	473.8	2.834e+004	1
6	826	464.5	4.749e+004	0
7	944	456.7	1.851e+004	0
8	1062	373.9	9167	0
9	1180	363.6	1.193e+004	0
10	1298	355.4	1.673e+005	0
11	1416	343.4	9409	0
12	1534	321.1	4334	0
13	1652	318.7	1.055e+005	0
14	1770	296.3	1.968e+008	0
15	1888	292.6	6.431e+004	0

16	2006	273.3	1.33e+005	0
17	2124	273.3	8.205e+004	1
18	2242	266.2	3.639e+005	0
19	2360	261.8	9775	0
20	2478	261.8	1.248e+005	1
21	2596	255.5	3.274e+005	0
22	2714	250.9	2.149e+005	0
23	2832	250.9	2.788e+005	1
24	2950	247.9	2.725e+005	0
25	3068	244.4	3.984e+005	0
26	3186	244.4	3.291e+004	1
27	3304	242.5	1.038e+005	0
28	3422	242.5	3378	1
29	3540	240.9	3169	0
30	3658	238	1.27e+008	0

t1 =

228.8626

Elapsed time is 1.071795 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	522	1.129e+004	2.129e+006	0
2	640	1.113e+004	1.101e+006	0
3	758	1.112e+004	1.548e+008	0
4	876	1.069e+004	1.547e+008	0

5	994	1.068e+004	8.473e+007	0
6	1112	1.063e+004	1.065e+006	0
7	1230	1.051e+004	8.174e+005	0
8	1348	1.051e+004	7.82e+007	1
9	1466	1.047e+004	7.824e+007	0
10	1584	1.046e+004	3.242e+007	0
11	1702	485.6	3.233e+007	0
12	1820	476.1	1.259e+006	0
13	1938	465.9	8.997e+005	0
14	2056	459	8.16e+005	0
15	2174	442.8	1.103e+004	0
16	2292	439.3	4218	0
17	2410	372.5	1.224e+004	0
18	2528	332.3	6487	0
19	2646	332.3	2561	1
20	2764	320.6	4336	0
21	2882	297.6	5804	0
22	3000	288.6	1.664e+005	0
23	3118	279.7	1.508e+005	0
24	3236	259.6	5.252e+004	0
25	3354	255.4	5.133e+004	0
26	3472	247.4	5.063e+004	0
27	3590	239.5	6.134e+004	0
28	3708	239.5	3334	1

29	3826	225.1	1.095e+004	0
30	3944	225.1	5844	1

t2 =

134.0567

Elapsed time is 4.434600 seconds.

Generation	f-count	Best f(x)	Mean f(x)	Stall Generations
1	236	1.074e+004	1.931e+005	0
2	354	1.065e+004	4.666e+006	0
3	472	1.055e+004	3.871e+005	0
4	590	1.053e+004	2.591e+005	0
5	708	1.042e+004	1.672e+004	0
6	826	1.039e+004	1.175e+004	0
7	944	1.038e+004	1.532e+004	0
8	1062	431.6	5.668e+004	0
9	1180	431.6	6.449e+004	1
10	1298	411.8	6.588e+005	0
11	1416	409.9	1.82e+004	0
12	1534	402.1	6333	0
13	1652	400.7	4596	0
14	1770	391.1	8850	0
15	1888	370.4	4327	0
16	2006	365.2	4460	0
17	2124	361.5	1.106e+004	0

18	2242	351.8	2.063e+004	0
19	2360	351.8	1.395e+004	1
20	2478	342.9	1868	0
21	2596	320.1	8916	0
22	2714	317.3	1662	0
23	2832	317.3	3118	1
24	2950	308.8	3729	0
25	3068	308.8	2680	1
26	3186	302.9	2758	0
27	3304	301.7	3422	0
28	3422	292.7	2200	0
29	3540	288.5	4.339e+004	0
30	3658	287.6	6965	0

t3 =

147.0744

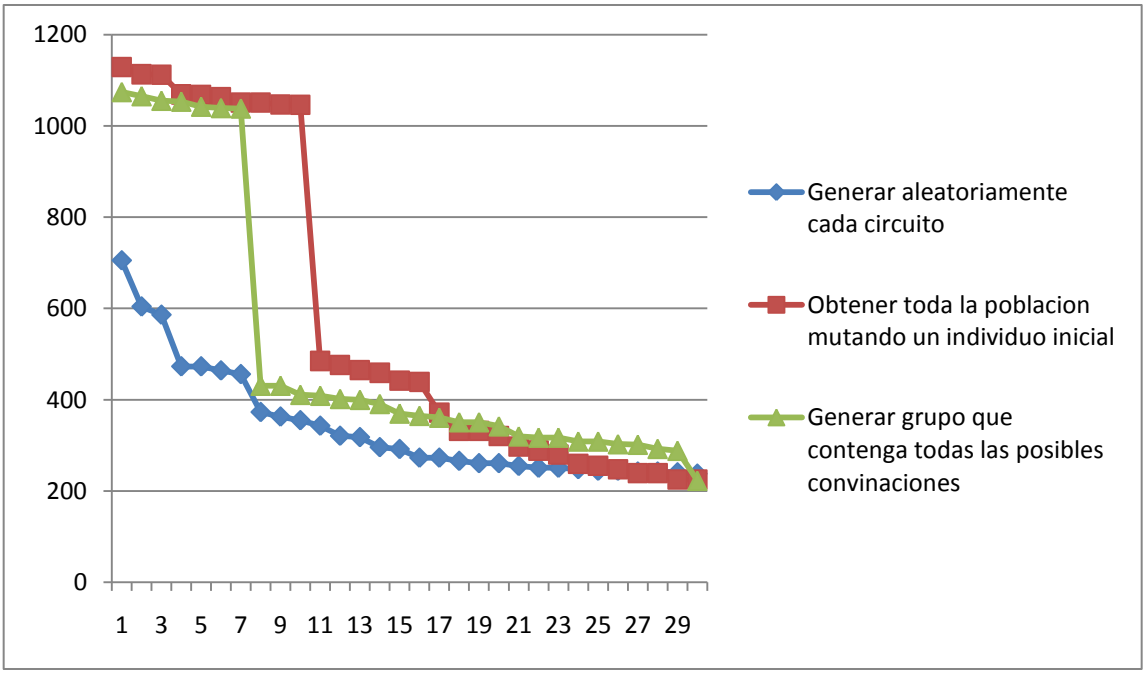


Figura 3.29: Comparación entre métodos de creación de población inicial para primeras 30 generaciones (circuito de 118 barras).

Ejemplo	1	2	3
Método usado			
Generar aleatoriamente cada circuito	84.95seg	85.65seg	86seg
Obtener toda la población mutando un individuo	1.032seg	1.05seg	1.07seg
Generar grupo que contenga todas las posibles combinaciones	4.234seg	4.336seg	4.43seg

Tabla 3: Comparación de métodos de creación de población

CONCLUSIONES Y RECOMENDACIONES

Como resultado del trabajo se obtienen las siguientes conclusiones:

1 Aunque la creación de la población por los métodos de crear la población a partir de la mutación de un individuo y crear población variada con todos los posibles interruptores abiertos es mucho más rápida, no garantizan una mayor variabilidad genética que el método de generar aleatoriamente cada individuo de la población inicial.

2 La correcta selección del número de individuos de la población de cada generación del AG optimiza el proceso de cálculo pues aminora el empleo de recursos computacionales (memoria y operaciones de cálculo) durante la búsqueda del individuo mejor adaptado.

3 La variabilidad genética de los individuos de la población obtenidos a partir del aumento del índice de mutación en la creación de la población inicial es vital en la obtención exitosa del circuito óptimo global.

Recomendaciones

1 Continuar con la busqueda de mejores métodos para minimizar el tiempo de obtención de resultados.

REFERENCIAS BIBLIOGRÁFICAS

[1] J. G. Díaz. (2012, Algoritmo Heurístico.
[2] J.A.Q.Gonzales;J.H.Kandjungulume, "Algoritmo Heurístico para la Reconfiguración de Sistemas de Distribución Mediante Intercambio de Ramas," pp. 196-204, 2012.
[3] G.Ochoa, "Introducción a la Computación Evolutiva y la Morfogénesis," 2005.
[4] J. A. G. Q. K.D.Ramirez. (2012, Reconfiguración de sistemas de distribución mediante algoritmos genéticos.
[5] J. A. D. I.J.Ramirez. (2006) New multiobjetive Tabu Search Algorithm for Fuzzy Optimal Planning of Power Distribution Sistems. 224-233.
[6] A.Marczyk. (2004, Algoritmos Geneticos y Computacion Evolutiva.
[7] Algoritmo de la Colonia de Hormigas [Online].
[8] A.M.C.Morales. (2010, Optimización de Enjambres de Partículas (PSO)-Método Heurístico.
[9] S. E. H. Randy Haupt (1998, Practical Genetic Algorithms.
[10] M. Gestal. (2010, Introducción a los Algoritmos Genéticos y la Programción Genética.
[11] A. R. C. Ballén. (2010). *Algoritmos Genéticos*. Available: http://www.slideshare.net/adrikano7/algoritmos-geneticos
[12] M. Mitchell. (1996, An Introduction to Genetic Algorithms.
[13] A. l. T. d. l. Fuente, "Algoritmos genéticos paralelos," 2005.
[14] E. Cantú-Paz, "Calculateurs Pralleles,Réseaux et Systems Répartis. "A survey of parallel genetic algorithms.´´," ed, 1998, pp. 141-171.
[15] A. A. García. (2012, Métodos de optimización: Algoritmos genéticos.
[16] P. T. Rodríguez-Piñero, "Introducción a los algoritmos genéticos y sus aplicaciones. ."
[17] MathWorks, Ed., *Global Optimization Toolbox User´s Guide*. 2015, p.^pp. Pages.
[18] R.C.Prim. (1957) Shortest connection networks and some generalisations. *Bell System Technical Journal*. 1389–1401.
[19] R. E. T. D. Cherition (1976) Finding minimum spanning trees. *SIAM Journal of Computing*. 724–741.
[20] C. E. L. Thomas H. Cormen, Ronald L. Rivest, Clifford Stein, "Introduction to Algorithms," *Second Edition. MIT Press and McGraw-Hill,The algorithms of Kruskal and Prim,* pp. 567–574, 2001.
[21] K. Ikeda. 21. Ejemplos usando JAVA (incluyen código) [Online]. Available: http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Prim.shtml
[22] Create and Solve Mazes by Kruskal's and Prim's algorithms [Online]. Available: http://www.cut-the-knot.org/Curriculum/Games/Mazes.shtml
[23] Animated example of Prim's algorithm [Online]. Available: http://students.ceid.upatras.gr/~papagel/project/prim.htm

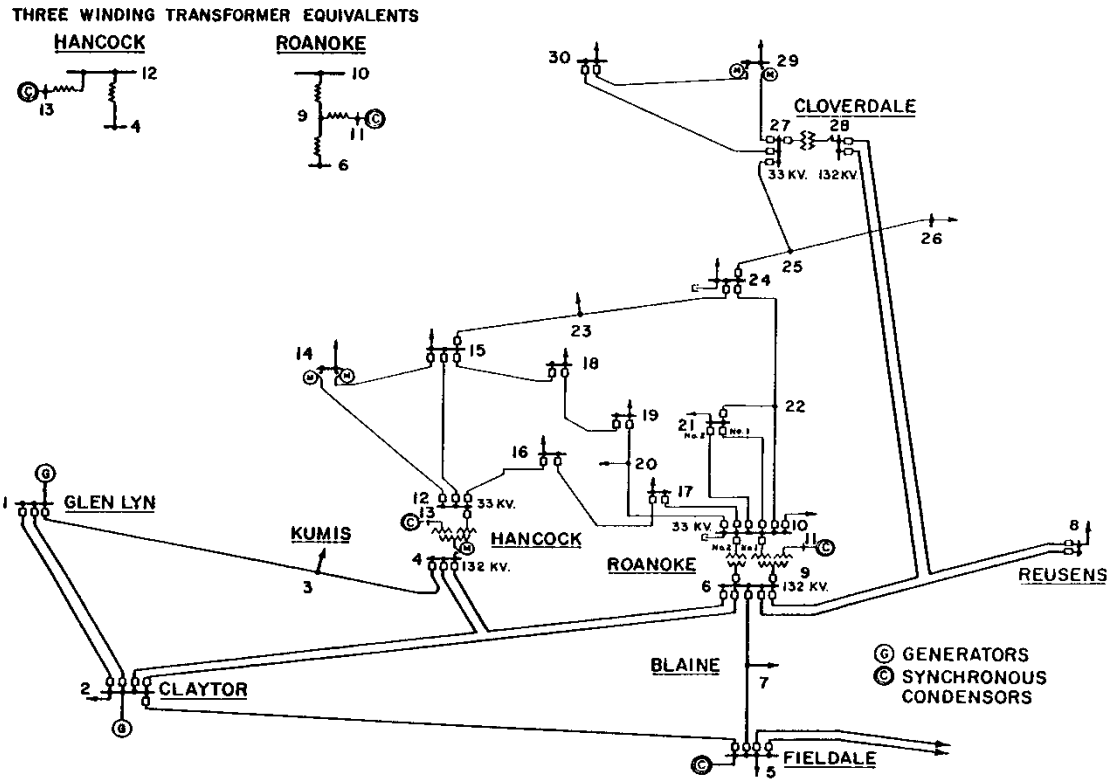
[24] Ejemplo interactivo (Java Applet) [Online]. Available: http://www.mincel.com/java/prim.html

[25] Ejemplo interactivo en español(Java Applet) [Online]. Available: http://www.dma.fi.upm.es/java/matematicadiscreta/

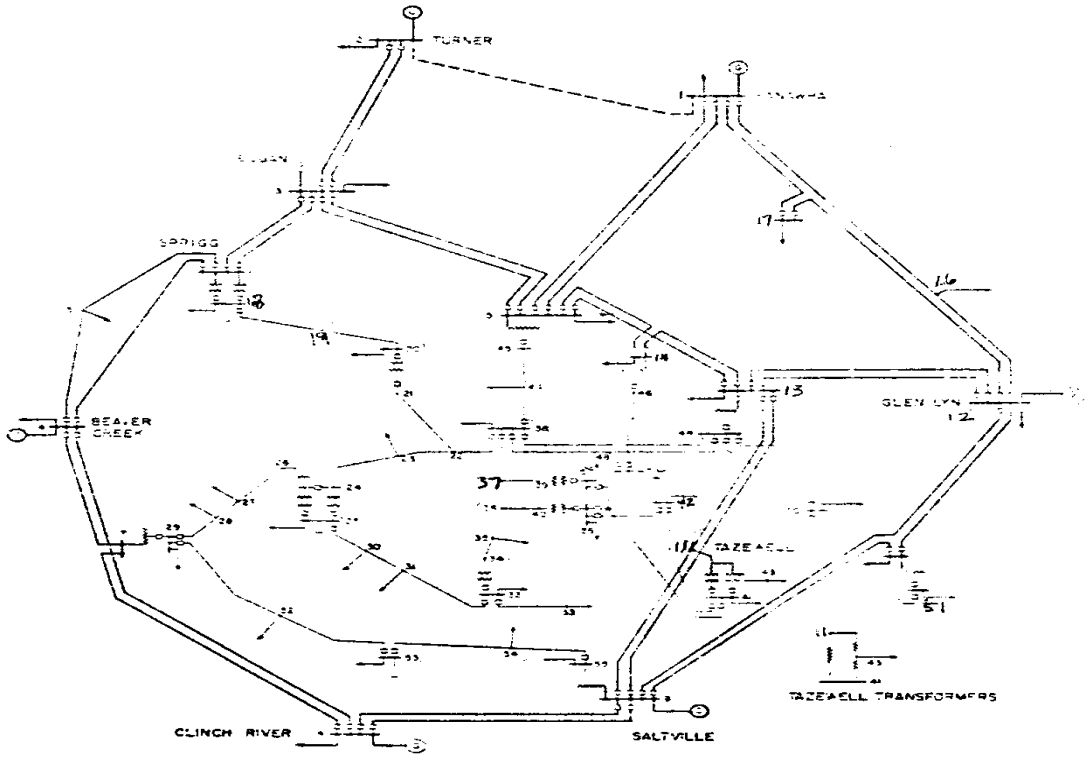
[26] Prim's algorithm code [Online]. Available: http://www.algorithm-code.com/wiki/Prim%27s_algorithm

ANEXOS

Anexo I Diagrama de circuito IEEE de 30 barras.



Anexo II Diagrama de circuito IEEE de 57 barras.



Anexo III Diagrama de circuito IEEE de 118 barras.